Oracle Application Server

LiveHTML および Perl アプリケーション開発者ガイド

リリース 4.0.8.2

2000 年 11 月

部品番号 : J01405-01

ORACLE®

Oracle Application Server LiveHTML および Perl アプリケーション開発者ガイド , リリース 4.0.8.2

部品番号 : J01405-01

原本名：Oracle Application Server Release 4.0.8.2 Developer's Guide: LiveHTML and Perl Applications

原本部品番号：A66960-04

原本著者：Kai Li

原本協力者：Sanjay Patil

目次

4 スクリプトの作成

5 Web Application Object を使用した開発

6 LiveHTML のトランザクション

7 Perl スクリプトからの CORBA オブジェクトへのアクセス

13 Perl スクリプトの作成

14 Perl インタプリタのアップグレード

15 トラブルシューティング

索引

はじめに

対象読者

このマニュアルは、Oracle Application Server の LiveHTML および Perl カートリッジを使用して Web アプリケーションを作成するユーザーを対象にしています。

Oracle Application Server のドキュメント・セット

この表に、Oracle Application Server のドキュメント・セットのリストを示します。

マニュアル名	部品番号
Oracle Application Server 概要	J01413-01
Oracle Application Server 管理者ガイド	J01403-01
Oracle Application Server セキュリティ・ガイド	J01411-01
Oracle Application Server パフォーマンス・チューニング・ガイド	J01414-01
Oracle Application Server PL/SQL アプリケーション開発者ガイド	J01404-01
Oracle Application Server JServlet および JSP アプリケーション開発者ガイド	J01408-01
Oracle Application Server LiveHTML および Perl アプリケーション開発者ガイド	J01405-01
Oracle Application Server EJB、ECO/Java および CORBA アプリケーション開発者ガイド	J01406-01
Oracle Application Server C++ CORBA アプリケーション開発者ガイド	J01407-01
Oracle Application Server PL/SQL Web Toolkit リファレンス	J01419-01
Oracle Application Server PL/SQL Web Toolkit クイック・リファレンス	J01420-01
Oracle Application Server JServlet Toolkit リファレンス	J01425-01

マニュアル名	部品番号
Oracle Application Server JServlet Toolkit クイック・リファレンス	J01426-01
Oracle Application Server カートリッジ・マネージメント・フレームワーク	J01412-01
Oracle Application Server エラー・メッセージ	J00103-01

マニュアルの表記規則

次の表に、このマニュアルで使用される表記規則を示します。

表記規則	例	説明
太字	**oas.h** **owsctl** **wrbcfg** **www.oracle.com**	ファイル名、 ユーティリティ、 プロセス、 および URL を表します。
斜体	*file1*	テキスト内の可変部分を表します。このプレースホルダを特定の値や文字列に置き換えます。
山カッコ	`<filename>`	コード内の可変部分を表します。このプレースホルダを特定の値や文字列に置き換えます。
固定幅フォント	`owsctl start wrb`	表示どおりに入力するテキスト。関数にも使用します。
大カッコ	`[-c string]`	オプション項目を表します。
	`[on|off]`	オプション項目の選択肢がそれぞれ垂直バー（|）で区切って示され、その中のいずれか 1 つを選択できます。
中カッコ	`{yes|no}`	必須項目の選択肢が垂直バー（|）で区切って示されます。
省略記号	n,...	その前の項目を何回でも繰り返すことができることを表します。

第 I 部

LiveHTML カートリッジ

1

LiveHTML カートリッジの概要

LiveHTML カートリッジにより、動的 HTML ページの生成が簡単になります。
動的に HTML ページを生成する場合、静的部分を含むページ全体を生成するためには、通常、スクリプトまたはプログラムを作成する必要があります。この場合、スクリプトやプログラムの作成およびそれぞれの動的 HTML ページの生成に時間がかかります。

LiveHTML には、動的 HTML ページを生成する別の方法があります。静的な HTML ページに Server-Side Includes（SSI コマンド）とスクリプトを埋め込むことにより、要求されるたびに HTML ページ全体を生成する必要がなくなります。

LiveHTML カートリッジは、SSI およびサーバー側スクリプトの Oracle Application Server におけるインプリメンテーションと拡張です。このカートリッジを使用すると、SSI コマンドとスクリプトを、直接 HTML ページに埋め込むことが可能です。

LiveHTML カートリッジには次のものが含まれます。

- Server-Side Includes（SSI）
- 埋込みスクリプト
- Web Application Object
- IDL-to-Perl コンパイラ
- プロセス・フロー

Server-Side Includes（SSI）

SSI を使用して、次の作業が可能です。

- 現行ファイルに他のファイルを挿入する。
- CGI 環境変数および SSI 環境変数を取得する。
- 現在の日付と時間を取得する。
- ファイルのサイズを取得する。

- ファイルの最終変更日付を取得する。
- スクリプトを実行する。
- 他のカートリッジにリクエストを送信する（標準 SSI の拡張）。Oracle Application Server の Inter-Cartridge Exchange サービス（ICX）を使用して、PL/SQL カートリッジや Java カートリッジなど、他のカートリッジにリクエストを送信したり、それらのカートリッジから返されるデータを HTML ページに挿入できます。

埋込みスクリプト

LiveHTML カートリッジは、HTML ページに埋め込まれたスクリプトを処理できます。このスクリプト機能により、静的 HTML とスクリプト言語のすべての機能を組み合せることが可能です。現在、このカートリッジのランタイムでは、スクリプト言語として Perl をサポートしています。

Perl の機能は、すべてユーザーのスクリプト内で使用可能です。たとえば、Perl の組込み関数のコール、独自の関数の定義と実行、スクリプトへの Perl モジュールの挿入（use 文を使用）、Perl の特殊変数の使用などが可能です。この詳細とスクリプトの例は、第 4 章「スクリプトの作成」を参照してください。

注意： LiveHTML カートリッジは、Perl カートリッジと同じインタプリタを使用します。このインタプリタのインプリメンテーションには、機能的に標準の Perl インタプリタと異なる部分があります。第 10 章の「標準版の Perl との相違点」の項を参照してください。

HTML ページ内のスクリプトから Web Application Object にアクセスすることもできま す。これらのオブジェクトは、HTML ページをアプリケーションとして管理および設計する場合に便利です。

スクリプト機能は、Microsoft Active Server Page と互換性があります。Perl で作成された Active Server Page がある場合、これを LiveHTML カートリッジで使用できます。

Web Application Object

LiveHTML ページのスクリプトから Web Application Object にアクセスできます。Web Application Object は、トランザクション型の Web ベース・アプリケーションの構築を強力にサポートできるように設計されたフレームワークです。詳細は、第 5 章「Web Application Object を使用した開発」を参照してください。

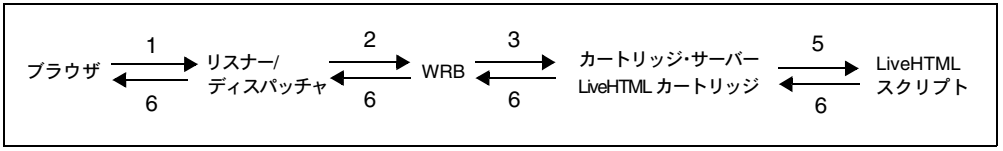
IDL-to-Perl コンパイラ

IDL-to-Perl コンパイラは、LiveHTML カートリッジに付属しています。このコンパイラ（**perlidlc**）により、CORBA オブジェクトへの Perl のバインドを生成し、LiveHTML ページの Perl スクリプトで CORBA オブジェクトが使用可能になります。詳細は、第 7 章「Perl スクリプトからの CORBA オブジェクトへのアクセス」を参照してください。

プロセス・フロー

次の図に、LiveHTML アプリケーションにおけるリクエストのフローを示します。

図 1-1 LiveHTML アプリケーションへのリクエストのプロセス・フロー



1. Oracle Application Server のリスナー・コンポーネントが、クライアントから LiveHTML カートリッジ宛てのリクエストを受け取ります。
2. ディスパッチャは、そのリクエストがカートリッジ用であることを確認し、これを WRB に転送します。
3. URL で指定されている仮想パスが LiveHTML カートリッジにマップされているため、WRB は URL を調べて、リクエストを LiveHTML カートリッジに送信します。
4. カートリッジ・サーバー・プロセスで実行されている LiveHTML カートリッジは、リクエストを受信し、その URL を調べて、解析する LiveHTML ファイルの名前を見つけます。
5. LiveHTML カートリッジは、ファイルをロードし、スクリプト・エンジンを実行して、ファイル内のスクリプトを解析します。
6. スクリプト・エンジンは、HTTP レスポンス・ヘッダーとボディの両方を含んだレスポンスを生成し、LiveHTML カートリッジに返します。LiveHTML カートリッジは、そのレスポンスを受信して、WRB にそれを返します。WRB は、リクエストを実行したクライアント・ブラウザにそのレスポンスを転送します。

注意： 現在、NT 環境では一部の環境変数（CLASSPATH、JAVA_HOME など）の拡張時の長さは 512 バイトに制限されています。
Oracle Application Server カートリッジの中には、環境変数が拡張されるものがあります。したがって、環境変数の長さが必ず 250-300 文字以内になるようにしてください。

2

アプリケーションの追加と実行

この章では、Oracle Application Server に LiveHTML アプリケーションを追加して、これをブラウザから実行する方法を説明します。

内容

LiveHTML アプリケーションの追加

Oracle Application Server アプリケーションを Oracle Application Server に追加するには、次の一般的なステップを実行します。

- アプリケーションを追加する。
- アプリケーションにカートリッジを追加する。

アプリケーションとカートリッジを追加するには、Oracle Application Server Manager に admin ユーザーでログインする必要があります。

アプリケーションとカートリッジを追加するには、次のステップを実行します。

1. ブラウザを起動し、Oracle Application Server の管理ページのトップ・ページを表示します。
2. サイト名の横の [+] をクリックして、そのサイトのコンポーネントを表示します。「Oracle Application Server」、「HTTP リスナー」、「アプリケーション」が表示されます。
3. 「アプリケーション」をクリックして、右フレームにアプリケーションを表示します。「アプリケーション」の横の [+] は、右フレームにアプリケーションを表示するかわりに、左フレームにサイトのアプリケーション・リストを表示するため、クリックしないでください。

4. 右フレームのアプリケーション・ページで、  をクリックします。「アプリケーションの追加」ダイアログ・ボックスが表示されます。
5. 「アプリケーションの追加」ダイアログ・ボックスで次の作業を行います。
   - 「アプリケーション・タイプ」:「LiveHTML」を選択します。
   - 「モードの設定」:「手動」を選択します。これにより、ダイアログ・ボックスを使用して設定データを入力できます。他のオプション「ファイルから」は、アプリケーションの設定データがすでにファイルに入力されていることを前提としています。
   - 「適用」をクリックします。

   「アプリケーションの追加」ダイアログ・ボックスが表示されます。
6. 「アプリケーションの追加」ダイアログ・ボックスで次の作業を行います。
   - 「アプリケーション名」: サーバーがアプリケーションを識別するために使用する名前を入力します。
   - 「表示名」: 管理用フォームで使用される名前を入力します。
   - 「アプリケーションのバージョン」: アプリケーションのバージョン番号を入力します。
   - 「適用」をクリックします。

     「適用」をクリックすると「成功」ダイアログ・ボックスが表示され、LiveHTML カートリッジをアプリケーションに追加できるボタンが表示されます。
7. 「成功」ダイアログ・ボックスで、「このアプリケーションにカートリッジを追加」ボタンをクリックします。「カートリッジの追加」ダイアログ・ボックスが表示されます。
8. 「カートリッジの追加」ダイアログ・ボックスで次の作業を行います。
   - 「カートリッジ名」: サーバーがアプリケーションの LiveHTML カートリッジを識別するために使用する名前を入力します。
   - 「表示名」: 管理用フォームで使用される名前を入力します。
   - 「仮想パス」: LiveHTML カートリッジを実行する場合に URL で指定する形式と同じ形式で、LiveHTML カートリッジのパスを入力します。このパスは、次に指定する物理パスにマップされます。この後の「「仮想パス」フォーム」の項を参照してください。
   - 「物理パス」: LiveHTML カートリッジ用のファイル（LiveHTML アプリケーション用のファイルを含む）の場所を示す物理ディレクトリ・パスを入力します。上で指定した仮想パスはこの物理パスにマップされます。

注意： セキュリティの理由で、最後に ".." が付く物理パスは指定できません。ただし、物理パスの設定で上位のディレクトリ・レベルを示すために、".." を使用することは可能です。たとえば、"/routines/../libraries/" などです。

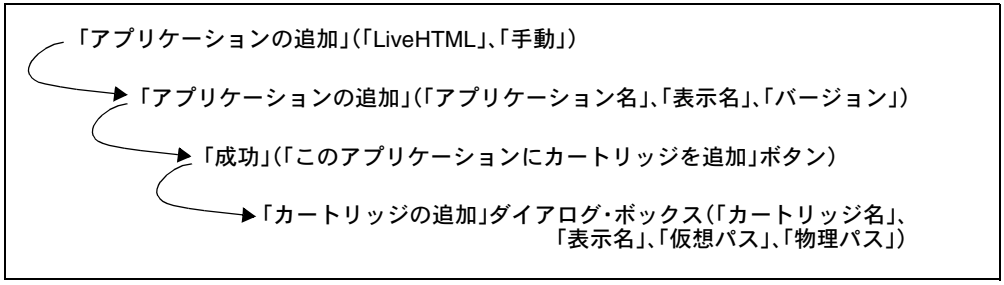
9. 「適用」をクリックします。
10. リスナーおよび Oracle Application Server の他のコンポーネントを停止して再起動します。

    詳細は、『Oracle Application Server 管理者ガイド』の「アプリケーションの管理」を参照してください。

注意： アプリケーションがナビゲーション・ツリーに表示されない場合は、[Shift] キーまたは [Ctrl] キーを押しながらブラウザの「リロード」ボタンをクリックします。

次の図は、入力が完了したダイアログ・ボックスのサマリーです。ダイアログ・ボックスのフィールドはカッコ内にリストされています。

図 2-1 LiveHTML アプリケーションとカートリッジを作成するためのダイアログ・ボックス



既存のアプリケーションへのカートリッジの追加

LiveHTML アプリケーションには、1 つ以上のカートリッジを持たせることが可能です。各カートリッジごとに異なるカートリッジ・パラメータの設定が必要な場合は、LiveHTML アプリケーションに複数のカートリッジを持たせる必要があります。たとえば、一部のカートリッジでは ICX を使用可能にし、他のカートリッジでは使用禁止にする場合などです。

カートリッジを LiveHTML アプリケーションに追加するには、次のステップを実行します。

1. ナビゲーション・ツリーで、カートリッジの追加先の LiveHTML アプリケーションの下の「カートリッジ」を選択します。

図 2-2 既存のアプリケーションへの LiveHTML カートリッジの追加

2.  をクリックして、「カートリッジの追加」ダイアログ・ボックスを表示します。
3. 「カートリッジの追加」ダイアログ・ボックスで次の作業を行います。
   - 「モードの設定」: 「手動」を選択します。
   - 「適用」をクリックすると「カートリッジの追加」ダイアログ・ボックスが表示されます。
4. 「カートリッジの追加」ダイアログ・ボックスで次の作業を行います。
   - 「カートリッジ名」: サーバーがアプリケーションの LiveHTML カートリッジを識別するために使用する名前を入力します。
   - 「表示名」: 管理用フォームで使用される名前を入力します。
   - 「仮想パス」: LiveHTML カートリッジを実行する場合に URL で指定する形式と同じ形式で、LiveHTML カートリッジのパスを入力します。このパスは、次に指定する物理パスにマップされます。この後の「「仮想パス」フォーム」の項を参照してください。
   - 「物理パス」: LiveHTML カートリッジ用のファイル（LiveHTML アプリケーション用のファイルを含む）の場所を示す物理ディレクトリ・パスを入力します。上で指定した仮想パスはこの物理パスにマップされます。

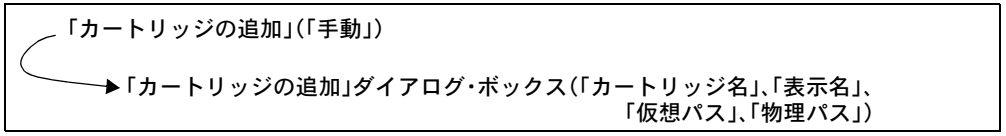
     注意： セキュリティの理由で、最後に ".." が付く物理パスは指定できません。ただし、物理パスの設定で上位のディレクトリ・レベルを示すために、".." を使用することは可能です。たとえば、"/routines/../libraries/" などです。

   - 「適用」をクリックします。

注意： 新しいアプリケーションがナビゲーション・ツリーに表示されない場合は、[Shift] キーまたは [Ctrl] キーを押しながらブラウザの「リロード」ボタンをクリックします。

次の図は、入力が完了したダイアログ・ボックスのサマリーです。ダイアログ・ボックスのフィールドはカッコ内にリストされています。

図 2-3 LiveHTML カートリッジを追加するためのダイアログ・ボックス



LiveHTML アプリケーションの設定

設定用フォームは、「アプリケーション」の「設定」と「カートリッジ」の「設定」の 2 つのセクションに分かれています。「アプリケーション」の「設定」セクションのフォームには、アプリケーション全体に適用されるパラメータが入っています。一方、「カートリッジ」の「設定」セクションのフォームには、特定のカートリッジにのみ適用されるパラメータが入っています。

「アプリケーション」の「設定」

「アプリケーション」の「設定」パラメータは、すべてのタイプのアプリケーションについて同じなので、「アプリケーションの設定」の章で説明します。

「カートリッジ」の「設定」

LiveHTML カートリッジの場合、「カートリッジ」の「設定」セクションには「仮想パス」フォームと「LiveHTML Parameters」フォームの 2 つのフォームが含まれます。

「仮想パス」フォーム

「仮想パス」フォームを使用して、LiveHTML カートリッジの仮想パスを指定できます。ユーザーは URL にこの仮想パスを指定して LiveHTML カートリッジを実行します。仮想パスは、そのアプリケーションの「Web の設定」ページにリストされているすべてのリスナーから使用可能です。

たとえば、**/myApp** という仮想パスを指定した場合、ユーザーは URL に **/myApp/***file* と入力することにより、LiveHTML アプリケーションを実行できます。*file* には **/myApp** 仮想パスに関連付けられている物理パスに存在するファイル名を指定します。

注意： セキュリティの理由で、最後に ".." が付く物理パスは指定できません。ただし、物理パスの設定で上位のディレクトリ・レベルを示すために、".." を使用することは可能です。たとえば、"/routines/../libraries/" などです。

「LiveHTML Parameters」フォーム

「LiveHTML Parameters」フォーム（図 2-4）で、LiveHTML カートリッジ固有のパラメータを設定できます。このフォームには次のパラメータが含まれます。

表 2-1 LiveHTML 固有の設定

オプション	説明
LiveHTML を使用可能にする	LiveHTML カートリッジを使用可能にするかどうか。 使用可能にしない場合、SSI コマンド、スクリプト・コマンドおよび Web Application Object は解釈されません。カートリッジの機能の一部のみ使用可能にする場合、LiveHTML を使用可能にして、不要な機能を使用禁止にできます。 **デフォルト**：使用可能
LiveHTML 拡張子のみ解析	「LiveHTML の拡張子」フィールドで指定された拡張子のファイルをカートリッジが解析するかどうか。 使用可能にした場合、カートリッジは「LiveHTML の拡張子」フィールドにリストされた拡張子のファイルのみ解析します。 使用可能にしない場合、カートリッジは拡張子に関係なく、すべてのファイルを解析します。 **デフォルト**：使用可能
LiveHTML の拡張子	カートリッジが処理するファイルの拡張子のリスト。 このフィールドは、「LiveHTML 拡張子のファイルのみ解析」フィールドが使用可能になっている場合のみ使用します。 カートリッジがすべての HTML ファイルを処理するように設定できます（つまり、拡張子のリストに "html" を加える）。ただし、すべての HTML ファイルが実際に SSI を使用している場合以外は、この設定を使用するとパフォーマンスが低下します。 **デフォルト**：`html shtml lhtml`
exec タグの処理を使用可能にする	カートリッジが SSI の exec コマンドを解析するかどうか。 **デフォルト**：使用可能

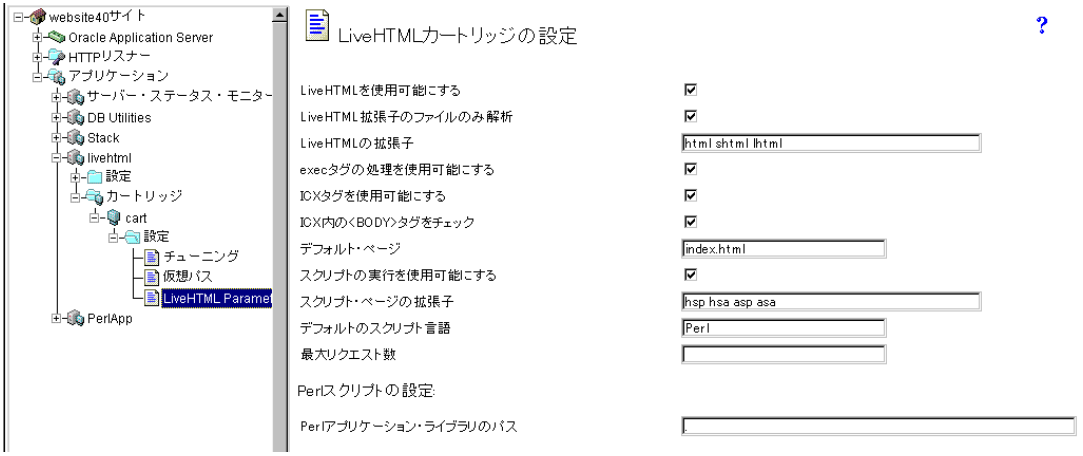
表 2-1 LiveHTML 固有の設定（続き）

オプション	説明
ICX タグを使用可能にする	カートリッジが request コマンドを解析するかどうか。 **デフォルト**：使用可能
ICX 内の <BODY> タグをチェック	カートリッジが ICX リクエストへのレスポンス内の <BODY> タグをチェックするかどうか。（ICX リクエストは request コマンドを使用して送信されます。） 使用可能にした場合、ICX レスポンスの <BODY> セクションのデータのみ、request コマンドを送信した HTML ページに挿入されます。ICX レスポンス内に <BODY> セクションが見つからない場合、カートリッジはエラーを発行します。 使用可能にしない場合、ICX レスポンス全体が HTML ページに含まれます。 **デフォルト**：使用可能
デフォルト・ページ	URL にファイルが指定されていない場合にクライアントに返される HTML ページ。 **デフォルト**：`index.html`
スクリプトの実行を使用可能にする	カートリッジがファイルに埋め込まれたスクリプトを解析するかどうか。 **デフォルト**：使用可能
スクリプト・ページの拡張子	カートリッジが埋込みスクリプトをチェックするファイル拡張子のリスト。 **デフォルト**：`hsp hsa asp asa`
デフォルトのスクリプト言語	デフォルトのスクリプト言語。現在、サポートされている言語は Perl のみです。ページ単位またはスクリプト・ブロック単位で異なる言語を指定できます。第 4 章の「スクリプト言語の指定」を参照してください。 **デフォルト**：`Perl`

表 2-1 LiveHTML 固有の設定（続き）

オプション	説明
最大リクエスト数	カートリッジ・サーバーが終了するまでに処理可能なリクエストの数。 このフィールドは、LiveHTML アプリケーションを開発する際に便利です。HTML ページから Perl ライブラリをコールすると、Perl インタプリタは Perl ライブラリをキャッシュに入れ、それ以降のリクエストにはそのキャッシュに入っているライブラリが使用されます。ライブラリを変更した場合、インタプリタによって新しいバージョンがロードされるようにする必要があります。これを行うには、現在のカートリッジ・サーバー・プロセスを終了し、新規のカートリッジ・サーバー・プロセスが（新規の Perl インタプリタを使用して）リクエストを処理できるようにします。これを行う簡単な方法は、「最大リクエスト数」の値を 1 に設定することです。 **デフォルト**: デフォルトはありません。つまり、カートリッジ・サーバーが処理可能なリクエストの数には上限がありません。
Perl アプリケーション・ライブラリのパス	Perl インタプリタが Perl ライブラリを検索するディレクトリ。 このオプションにパスを追加する場合、フルパス名を使用する必要があります。複数のディレクトリを指定する場合は、":" でそれぞれのディレクトリを区切ります。 **デフォルト**: .（カートリッジ・サーバー・プロセスの現行作業ディレクトリ）

図 2-4 「LiveHTML カートリッジの設定」フォーム



セキュリティ

ユーザーがサーバー上でスクリプトを実行できるようにすると、セキュリティの問題やその他のリスクが生じます。セキュリティ・リスクを最小にするために、次の方法で LiveHTML カートリッジを設定できます。

- 実行スクリプトが含まれる SSI コマンドを実行しないように、カートリッジを設定できます。カートリッジは環境変数の値を取得する、または他のファイルを挿入することのみ可能です。これは、exec タグを使用禁止にすることによって行います。
- また、他のカートリッジを実行するリクエストを実行しないように設定できます。これは、ICX タグを使用禁止にすることによって行います。
- さらに厳しくする場合は、カートリッジ全体を使用禁止にし、SSI コマンドまたはスクリプトがまったく解析されないようにします。これは、LiveHTML オプションを使用禁止にすることによって行います。
- 認証方式および制限方式を使用して LiveHTML カートリッジの仮想パスを保護できます。詳細は、『Oracle Application Server セキュリティ・ガイド』を参照してください。

3

Server-Side Includes の使用

内容

SSI コマンド

Server-Side Includes（SSI）コマンドは、HTML のコメントの書式を使用して設定され、次の書式を使用します。

```
<!--#command [param1="value1" param2="value2" ...] -->
```

詳細は次のとおりです。

- *command* は SSI コマンドの名前
- *param1* と *param2* はコマンドに渡すパラメータの名前
- *value1* と *value2* はパラメータの値

パラメータはコマンドによって異なります。パラメータを使用しないコマンドもあります。

LiveHTML ファイルでは、1 行に 1 つの SSI コマンドのみ含めることが可能です。たとえば、次のようになります。

```
<p><!--#include file="notes.htm"-->  Server name: <!--#echo var="SERVER_NAME" -->
```

は、2 つの行に分ける必要があります。

```
<p><!--#include file="notes.htm"-->
Server name: <!--#echo var="SERVER_NAME" -->
```

出力は同じです。

エラー

SSI コマンドの処理中にエラーが発生した場合（たとえば、指定されたコマンドが見つからない、解析エラーが発生したなど）、"Server Side Processing Error" というメッセージが表示されます。

`config errmsg` コマンドを使用して、このエラー・メッセージを変更できます。たとえば、次のコマンドにより、エラー・メッセージが "If you see this message, an error occurred" に設定されます。

```
<!--#config errmsg="If you see this message, an error occurred.">
```

カスタム・メッセージは、すべてのエラーに使用されます。

特殊文字

LiveHTML カートリッジでは、$ と一重引用符には特別な意味があります。これらの文字を SSI タグの中で文字どおりの意味で指定する必要がある場合、文字の前に円記号（¥）を付けます。

コマンド・サマリー

次の表に、LiveHTML カートリッジによって処理可能な SSI コマンドを示します。

表 3-1 SSI コマンド

コマンド	説明
config	インクルード・ファイルまたはスクリプトの解析方法をパラメータに設定します。通常、このコマンドがファイル内で 1 番最初に記述される LiveHTML コマンドです。
include	生成された HTML ページのこの位置に、ファイルをインクルードするよう指定します。

表 3-1 SSI コマンド（続き）

コマンド	説明
echo	環境変数の値を出力します。
fsize	ファイルのサイズを出力します。
flastmod	ファイルの最終更新日付を出力します。
exec	スクリプトを実行します。
request	Inter-Cartridge Exchange（ICX）を使用して、他のカートリッジにリクエストを送信します。

config

ファイル内の他の SSI コマンドのフォーマット情報を定義します。通常、config コマンドはファイルの一番最初に使用する SSI コマンドです。1 つのファイルで複数の config コマンドを使用できます。

パラメータ	説明
errmsg	ドキュメントの解析中にエラーが発生した場合にクライアントに送信されるエラー・メッセージ。
timefmt	日付の表示に使用するフォーマット。この表記規則はライブラリ・コール strftime で使用されます。（詳細は、strftime の man ページを参照してください。） そのフォーマット内の通常の文字は置換されずにドキュメントにコピーされるため、"on" や "at" など、必要な文字列を挿入できます。
sizefmt	ファイル・サイズの表示に使用するフォーマット。使用できる値は次のとおりです。 ■ bytes - ファイル・サイズをバイトで示します。 ■ abbrev - ファイル・サイズを KB または MB で示します。
cmdecho	これより後に実行される CGI 以外のスクリプトの出力を、この HTML ページに取り込むかどうか。使用できる値は ON と OFF です。ON は出力が組み込まれることを示します。デフォルトは OFF です。
cmdprefix	スクリプトの出力の各行の先頭に追加する文字列。
cmdpostfix	スクリプトの出力の各行の最後に追加する文字列。

例：

```
<!--#config errmsg="A parse error occurred" sizefmt="bytes" cmdecho="ON"-->
```

include

生成された HTML ページにファイルを挿入します。挿入できるファイルは、別の LiveHTML ファイルまたは ASCII ファイルです。Oracle Application Server は、挿入されるファイルを拡張子で識別します。

パラメータ	説明
virtual	ファイルの仮想パス。仮想パスのディレクトリ・マッピングは、Oracle Application Server の管理者が Oracle Application Server Manager を使用して設定します。
file	現行ディレクトリからの相対パス名。上位ディレクトリの参照や絶対パス名は使用できません。

インクルード・ファイルが完全な HTML ドキュメントである場合、LiveHTML カートリッジはドキュメントの <BODY> セクションのデータのみ読み込みます。ドキュメントの <HEAD> セクションのデータは挿入されません。

例 :

リンクを指定するファイルを挿入することにより、一連の共通リンクを挿入できます。
次のコマンドを HTML ファイルに挿入するとします。

```
<!--#include file="links.html"-->
```

そして、**links.html** ファイルに次の行が含まれるとします。

```
<p><a href="home.html">Home</a> |
<a href="index.html">Index</a> |
<a href="search.html">Search</a>
```

この結果、次のリンクが Web ページに挿入されます。

```
Home | Index | Search
```

echo

標準の CGI 環境変数または SSI 環境変数の値を出力します。

パラメータ	説明
var	環境変数の名前。

CGI 環境変数のリストは、**http://hoohoo.ncsa.uiuc.edu/cgi-1.1/** を参照してください。

次の表に、SSI 環境変数を示します。

表 3-2 SSI 環境変数

SSI 環境変数	説明
DOCUMENT_NAME	現行のファイル名。
DOCUMENT_URI	このファイルの仮想パス。
QUERY_STRING_UNESCAPED	クライアントが問合せ文字列を送信すると、これはエスケープを使用しないバージョンであるため、シェルの特殊文字はすべて円記号（¥）に置き換わります。
DATE_LOCAL	現在日付とローカル・タイム・ゾーン。最新の config timefmt コマンドで指定されたフォーマットで表示されます。
DATE_GMT	グリニッジ標準時による現在日付とタイム・ゾーン。最新の config timefmt コマンドで指定されたフォーマットで表示されます。
LAST_MODIFIED	ファイルの最終更新日付。最新の config timefmt コマンドで指定されたフォーマットで表示されます。

例 :

コマンド

```
Current date/time: <!--#echo var="DATE_LOCAL"-->
```

により、次の行が生成されます。

```
Current date/time: Thursday, April 17, 1997 03:15 PM
```

fsize

ファイル・サイズ。最新の "config sizefmt" コマンドで指定されたフォーマットで表示されます。

このコマンドは、include と同じパラメータを使用します。

例 :

コマンド

```
<!--#fsize file="logo.jpg"-->
```

により、logo.jpg ファイルのサイズが生成されます。

このコマンドの一般的な使用方法の 1 つは、ダウンロードする画像ファイルのサイズをユーザーに提供することです。これにより、ファイル・サイズを調べてそれを手動で入力する必要がなくなるため、ダウンロード可能なイメージを頻繁に追加および変更する場合に大幅に時間を節約できます。

flastmod

ファイルの最終更新日付。最新の "config timefmt" コマンドで指定されたフォーマットで表示されます。

このコマンドは、include と同じパラメータを使用します。

例 :

コマンド

```
<!--#flastmod file="releases.html"-->
```

により、releases.html ファイルの最終変更日が生成されます。

exec

スクリプトを実行します。パラメータで、そのスクリプトが CGI かどうかを指定します。

exec コマンドを使用する前に、「LiveHTML カートリッジの設定」でこのコマンドを使用可能にしておく必要があるので注意してください。「LiveHTML カートリッジの設定」ページで、「exec タグの処理を使用可能にする」オプションをチェックします。

パラメータ	説明
cmd	CGI 以外のスクリプトを指定します。実行はオペレーティング・システムに渡され、指定した文字列が、コマンドライン・インタフェースから入力された場合と同じように解析されます。スクリプトはフルパスで指定する必要があります。 SSI 環境変数を参照可能です。 **注意 :** HTML ページにスクリプトの出力を挿入するには、その HTML ページに次のコマンドを設定する必要があります。 `<!--#config cmdecho="ON"-->`
cgi	CGI スクリプトを指定します。値は該当する CGI スクリプトの仮想パスです。URL の場所は、自動的に HTML アンカーに置換されます。

例 :

```
<!--#config cmdecho="ON" -->
...
<!--#exec cmd="/bin/who" -->
```

request

ICX（Inter-Cartridge Exchange）リクエストを作成します。Oracle Application Server の ICX 機能を使用して HTTP リクエストを作成することにより、カートリッジ間の通信を行うことができます。たとえば、このコマンドを使用してリクエストを PL/SQL カートリッジに送信し、データベース内のストアド・プロシージャを実行したり、ストアド・プロシージャの結果を LiveHTML ドキュメントに埋め込むことが可能です。

`request` コマンドは、Oracle Application Server の SSI コマンド・セットの拡張です。

注意： `request` コマンドを使用する前に、「LiveHTML カートリッジの設定」でこのコマンドを使用可能にしておく必要があります。「LiveHTML カートリッジの設定」ページで、「ICX タグを使用可能にする」をチェックします。

パラメータ	説明
`url`	リクエストを送信する URL。URL の構文は次のとおりです。 http://*user*:*password*@*host*:*port*/*url-path*?*QS* 詳細は次のとおりです。 *url-path* は一般的な URL の使用方法を拡張しています。これは、LiveHTML カートリッジが変数の置換をサポートしているためです。 *QS* は名前と値のペアの形式の問合せ文字列です。この構文は、GET 方式の問合せ文字列と同じです（名前と値のペアが & 文字で区切られ、スペースが + 文字に置換されます）。 問合せ文字列を一重引用符で囲むと、LiveHTML カートリッジにその文字列をエンコードできます。

問合せ文字列の値の使用

問合せ文字列の名前と値のペアを使用して LiveHTML ページを実行して、そのページで request コマンドの値を参照できます。この方法を使用すると、request コマンド内で動的 URL を生成できます。

たとえば、LiveHTML ページが **showUserProp.shtml** で、ユーザーの値とプロパティの値が必要な場合、次のようにしてそのページを実行できます。

```
http://domain/showUserProp.shtml?user=chris&property=job+title
```

この URL は、ユーザーがフォームのフィールドにユーザー名とプロパティを入力可能な HTML フォームによって、自動的に生成されます。

showUserProp.shtml ページの request コマンドでは、変数名の前に $ を付けることによって、user 変数と property 変数を参照できます。この例では、変数は $user と $property です。

showUserProp.shtml ページの request コマンドは、次のようになります。

```
<!--#request URL="/$user/work?SQLString=
                'select $property from employee'&name=$user" -->
```

LiveHTML カートリッジは、前述のリクエストを次のように展開します。

```
/chris/work?SQLString=select+job%20title+from+employee&name=chris
```

一重引用符内の内容は LiveHTML がエンコードするため、エンコードされていない URL とみなされます。HTTP の URL の残り部分は、正しくエンコードする必要があります。

SSI の例

- 日付と時間の表示
- 現行ファイルに関する情報の取得
- 他のファイルに関する情報の取得
- ブラウザ情報の表示
- ホストとサーバーの情報の出力
- データベースへのアクセス

日付と時間の表示

次の config コマンドで、echo コマンドで使用される日付と時間のフォーマットを定義しています。

```
<!--#config timefmt="%A, %B %d, %Y, at %I:%M %p"-->
<p>GMT date/time is
<!--#echo var="DATE_GMT"-->
<p>LOCAL date/time is
<!--#echo var="DATE_LOCAL"-->
<p>Updated on
<!--#echo var="LAST_MODIFIED"-->
```

これにより、次の行が生成されます。

```
GMT date/time is Friday, August 23, 1996, at 03:14 AM
LOCAL date/time is Thursday, August 22, 1996, at 08:14 PM
Updated on Tuesday, August 13, 1996, at 03:42 AM
```

現行ファイルに関する情報の取得

次の行を入力すると

```
This document is <!--#echo var="PATH_TRANSLATED"-->
Its virtual path is <!--#echo var="DOCUMENT_URI"-->
```

UNIX では、次のように表示されます。

```
This document is /oracle/test/livehtml/sstest.html
Its virtual path is /sample/livehtml/sstest.html
```

NT では、次のように表示されます。

```
This document is ¥oracle¥test¥livehtml¥sstest.html
Its virtual path is ¥sample¥livehtml¥sstest.html
```

他のファイルに関する情報の取得

fsize コマンドと flastmod コマンドにより、現行ドキュメントだけではなく、サーバー上のファイルについても、ファイル・サイズおよび最終更新日を取得できます。

たとえば、次の行

```
<!--#config sizefmt="bytes"-->
<p>This gives the file size of 'sstest.html' in bytes:
<!--#fsize file="sstest.html"-->
```

により、次の情報が生成されます。

```
This gives the file size of 'sstest.html' in bytes: 6405 bytes
```

次の行

```
<!--#config sizefmt="abbrev"-->
<p>This gives the file size of 'sstest.html' in bytes:
<!--#fsize file="sstest.html"-->
```

により、次の行が生成されます。

```
This gives the file size of 'sstest.html' in kilobytes: 6 Kbytes
```

ブラウザ情報の表示

ユーザーがそのページの表示に使用しているブラウザとそのバージョンを、ユーザーに表示できます。

次の行

```
<p>You are using <!--#echo var="HTTP_USER_AGENT" -->
```

により、次の行が生成されます。

```
You are using Mozilla/4.02 [en] (X11; I; SunOS 5.5.1 sun4u)
```

ホストとサーバーの情報の出力

次の行

```
<p>Host: <!--#echo var="REMOTE_HOST" -->
(<!--#echo var="REMOTE_ADDR" -->)
<p>Server: <!--#echo var="SERVER_NAME" -->
(<!--#echo var="SERVER_SOFTWARE" -->)
```

により、次の情報が生成されます。

```
Host: test.us.oracle.com (123.45.67.89)
Server: test.us.oracle.com (Oracle_Web_Listener/4.0.6.3.0EnterpriseEdition)
```

データベースへのアクセス

次のコマンドは、ICX を使用して PL/SQL カートリッジを実行し、**myproc** ストアド・プロシージャを実行します。ストアド・プロシージャの結果は、LiveHTML ページに挿入されます。この例では、**/db/plsql/** 仮想パスが PL/SQL カートリッジに関連付けられていることを前提としています。

```
<!--#request url="/db/plsql/myproc"-->
```

4

スクリプトの作成

LiveHTML カートリッジのスクリプト機能により、HTML ページ内にスクリプトおよびフロー制御タグを埋め込むことができます。スクリプトは、標準の HTML タグの間に挿入できます。この構成により、Server-Side Includes（SSI）がサポートしているコマンド以上に複雑なコマンドを実行できます。たとえば、Web アプリケーションのユーザーを判断するスクリプトを作成し、そのユーザー名を使用してその HTML ページの残りの部分の内容を決定できます。フロー制御タグ（"if-then" 文）により、式が True である場合のみスクリプトを実行したり、クライアントにそのページ上の HTML データを返すことができます。たとえば、ユーザーが使用しているブラウザをチェックして、そのタイプによって異なる HTML データを返すことなどが可能です。

スクリプト機能を使用して、LiveHTML ページから Web Application Object にアクセスできます。Web Application Object は、トランザクション型および非トランザクション型の Web アプリケーションを構築するためのランタイム・サービスを使用したフレームワークを提供します。詳細は、第 5 章「Web Application Object を使用した開発」を参照してください。

現在、Perl インタプリタを持つカートリッジ・ランタイムは、スクリプト言語として Perl バージョン 5.004_01 をサポートしています。**http://www.perl.org** から、Perl 言語に関する最新情報を入手できます。また、LiveHTML Perl インタプリタ用の、多くの便利なモジュールを **http://www.perl.com/CPAN/modules/00modlist.long.html** からダウンロードできます。これらのモジュールにより、スクリプトの機能を拡張することが可能です。Perl インタプリタをアップグレードする場合は、第 14 章「Perl インタプリタのアップグレード」を参照してください。

注意： LiveHTML Perl インタプリタは、Perl DBD::Oracle ドライバおよび DBI API を使用することにより、データベースの使用が可能です。Oracle データベースの問合せの実例については、5-8 ページの「Oracle データベースのデータの問合せと取出し」を参照してください。

内容

スクリプトのファイル名拡張子

スクリプトが入っている LiveHTML カートリッジ用の HTML ページは、HTML スクリプト・ページ（HSP）と呼ばれます。スクリプト・ページのデフォルトの拡張子は `hsp`、`hsa`、`asp` および `asa` です。

Oracle Application Server Manager の「LiveHTML Parameters」フォームの「スクリプト・ページの拡張子」フィールドに追加すれば、他の拡張子も使用できます。

スクリプト機能の使用可能と使用禁止

スクリプト機能を使用可能または使用禁止にできます。LiveHTML カートリッジの「LiveHTML Parameters」ページ（Oracle Applications Server Manager にある）で、「スクリプトの実行を使用可能にする」ボックスにチェックを付けるか、またはチェックを外します。

チェックを外すと、カートリッジは「スクリプト・ページの拡張子」で指定された拡張子のファイルに含まれているスクリプトを処理しません。カートリッジは、これらのファイルの解析も行いません。このボックスにチェックが付いている場合、これらのファイルのスクリプトは処理されます。

スクリプト言語の指定

LiveHTML ページにスクリプト言語を指定する方法は 3 つあります。これらは、それぞれ適用範囲が異なります。

全般的なデフォルト言語

LiveHTML カートリッジに属している HTML ページのデフォルトのスクリプト言語は、すべて Oracle Application Server Manager の「LiveHTML カートリッジの設定」ページで指定します（第 2 章の「「カートリッジ」の「設定」」を参照）。このデフォルト言語は、次の 2

つの項で説明する方法を使用して別の言語を指定しない限り、すべてのスクリプトに適用されます。

特定のページの指定

特定のページに別のスクリプト言語を使用する場合、<% @Language=*language* %> タグを使用して言語を指定できます。指定された言語は、そのページの <%...%> タグと <%=...%> タグの範囲内にあるスクリプトに適用されます。<% @Language=*language* %> がページ内に存在しない場合、<%...%> タグと <%=...%> タグで囲まれたスクリプトには、「LiveHTML カートリッジの設定」フォームで指定されたデフォルト言語が適用されます。

ページ内のスクリプト・ブロックの指定

特定の状況で、スクリプトのブロックに Oracle Application Server Manager または <% @Language=*language* %> で指定されたデフォルト以外の言語を使用する場合があります。このような場合、<SCRIPT>...</SCRIPT> タグの LANGUAGE 属性を使用します。詳細は、次の項を参照してください。

注意： 現在のリリースでは、"Perl" のみサポートされています。将来のリリースで他の言語もサポートされる予定です。また、LiveHTML カートリッジ・サーバー・プロセスで使用される Perl インタプリタは、そのプロセスの中で 1 回だけインスタンスが生成されます。LiveHTML アプリケーションで予期しない結果が発生しないようにするために、LiveHTML スクリプトの中で使用する変数は、すべて常に初期化するようにしてください。

スクリプトの埋込み

LiveHTML ページにスクリプトを埋め込むためのタグはいくつかあります。次のとおりです。

- <%...%>
- <%= ... %>
- <SCRIPT>...</SCRIPT>

次の表に、それぞれのタグをどのような場合に使用するかをまとめます。

表 4-1 スクリプト・タグの使用方法一覧

タグ	使用する場合
<%...%>	このタグに囲まれたスクリプトがそのページのデフォルトの言語で作成されている場合に使用。
<%=...%>	式の結果を LiveHTML ページの HTML 出力の一部として出力する必要がある場合に使用。

表 4-1 スクリプト・タグの使用方法一覧（続き）

タグ	使用する場合
<SCRIPT>...</SCRIPT>	このタグに囲まれたスクリプト・ブロックがそのページのデフォルト言語で作成されていない場合に使用。そのスクリプト・ブロックで使用されている言語は、このブロックにのみ適用されます。

注意： <SCRIPT>...</SCRIPT> タグまたは <% ... %> タグのスクリプトの結果は、スクリプトが Response オブジェクトの write メソッドを通じて書込みを行なった場合のみ、ユーザーが参照できます。たとえば、Perl の場合、その構文は $Response->write("text"); になります。

<%...%>

<%...%> タグは、各 LiveHTML ページの <% @Language = *language* %> タグで指定された言語のスクリプトを囲みます。このタグがページ内に存在しない場合、スクリプトには「LiveHTML カートリッジの設定」フォームで指定されたデフォルト言語が適用されます。制御構造を使用して標準の HTML タグにこれらのタグを組み合せ、どの HTML タグをクライアントに送信するかを決定できます。これらのタグを他の HTML タグの中に埋め込むこともできます。これは、動的に生成されるリンク（A タグの HREF 属性）に便利です。

構文

```
<%
script
%>
```

例

次の例は、"String to print" というテキストを、フォント・サイズを 3 ～ 7 まで増やして出力します。

```
<%
  $str = "String to print";
  for ($fontsize = 3; $fontsize < 8; $fontsize++) {
%>
    <font size = <% $Response->write($fontsize) %> >
    <p> <% $Response->write($str) %> </font>
<% } %>
```

注意： 変数の有効範囲は 1 つのスクリプト・ブロックに限定されません。変数は、その HTML ページ全体の複数のスクリプト・ブロックに適用されます。

次の例で、if-then 文を使用して、クライアントに送信する HTML 文を判断する方法を示します。

```
<%
  if ($something_failed) {
%>
<p>An error occurred while processing your request.
<% } else { %>
<p>Here are the results of your request.
<% } %>
```

<%= ... %>

<%= *expression* %> タグは、<% @Language=*language* %> タグで指定された言語で作成されている、指定された式の値を表示します。

構文

```
<%= expression %>
```

例

次のスクリプトにより、"10 ...9 ...8 ...7 ...6 ...5 ...4 ...3 ...2 ...1 ... Fire!" と出力されます。

```
<% for ($countdown = 10; $countdown > 0; $countdown--) { %>
<%= $countdown %> ...
<% } %>
Fire!
```

<SCRIPT>...</SCRIPT>

このタグを使用して、このタグに囲まれているスクリプト・ブロックに使用する特定の言語を指定できます。指定する言語は、<% @Language=*language* %> タグまたはカートリッジの設定フォームで指定された言語とは異なる言語でも構いません。

構文

```
<SCRIPT
    RUNAT=SERVER
    [LANGUAGE=language]>
script
</SCRIPT>
```

属性

RUNAT - このタグを、クライアントではなく、LiveHTML カートリッジで処理するように指定します。この属性は、スクリプトを処理する LiveHTML カートリッジの場合は必須です。この属性が指定されていない場合、このタグはクライアント・ブラウザによって処理されます。現在、使用可能な値は SERVER のみです。

LANGUAGE - スクリプトの言語を指定します。現在、使用可能な値は "Perl" のみです。この属性が指定されていない場合、「LiveHTML カートリッジの設定」フォームで指定されたデフォルト言語が使用されます。

例

次の例は、thank you というメッセージを表示します。

```
<SCRIPT RUNAT=SERVER LANGUAGE=Perl>
$Response->write("Thank you for using our <em>interactive web application
                </em>.¥n");
</SCRIPT>
```

スクリプト内での CORBA オブジェクトの使用

<OBJECT> タグは、現行の HTML ページ内のスクリプトで CORBA オブジェクトが使用されることを宣言します。このタグを使用して、同じ HTML ページ内で複数のオブジェクトを宣言することが可能ですが、それぞれの宣言の有効範囲はその HTML ページ内のみです。オブジェクトは、指定された ID を使用してスクリプト内で参照可能です。

構文

```
<OBJECT
    RUNAT = "SERVER"
    ID= identifier
    OR = obj_ref_string | CARTRIDGE = cartridge_identifier>
```

属性

RUNAT - このタグを、クライアントではなく、LiveHTML カートリッジで処理するように指定します。この属性は、LiveHTML カートリッジがオブジェクトを宣言する場合は必須です。この属性が指定されていない場合、このタグはクライアント・ブラウザによって処理されます。現在、使用可能な値は SERVER のみです。

ID - スクリプトでオブジェクトのインスタンスを参照するために使用する名前を指定します。この属性は必須です。

オブジェクトを探すために、OR または CARTRIDGE のいずれかの属性が必要です。

OR - 文字列の形式でオブジェクト・リファレンスを指定します。

CARTRIDGE - オブジェクトを定義してインスタンス生成する CORBA カートリッジの名前を指定します。カートリッジは、リソース・マネージャによって管理されます。

例

次の例は、<OBJECT> タグで定義されている Menu オブジェクトから HREF 属性の値を判断します。

```
<OBJECT
    RUNAT = "SERVER"
    ID = "Menu"
    CARTRIDGE = "gui_set/menu_obj">
...
<A HREF="<%= $Menu->GetRef($date) %>">Daily Menu</A>
<!-- GetRef() is a method in the $Menu object -->
<!-- $date contains a date that the user could have entered earlier -->
```

次の例は、LiveHTML ページに埋め込まれた Perl スクリプトで使用する 3 つの CORBA オブジェクトを、<OBJECT> タグを使用して定義する方法を示しています。各オブジェクトは、OR または CARTRIDGE を使用して参照します。

```
<!-- The embedded Perl scripts below retrieve balances from three accounts belonging
to the same person using three CORBA object "account managers". -->

<HTML>
<% $AccName = "Robert"; %>
<TITLE><%= $AccName %>'s Account Information</TITLE>
<BODY>

<!-- CORBA object referenced using object reference (for checking information). -->
<OBJECT RUNAT="SERVER" ID="AccountManagerChecking"
OR="IOR:000000000000001C49444C3A42616E6B2F4163636F756E744D616E616765723A312E300
00000000100000000000009C000101000000000A7270616E672D73756E00158800000080000100
009C82B05A000000000000003200000000000000000031E331663292300000000010059000000
00A7270616E672D73756E0000000000000631323338390000000000000000000050000001C4944
4C3A42616E6B2F4163636F756E744D616E616765723A312E30000000000A7069643A31323338390
000000000000000000000">

<!-- Checking account balance. -->
<% $AccountChecking = $AccountManagerChecking->open($AccName); %>
Checking account balance is $<%= $AccountChecking->balance() %><P>

<!-- CORBA object referenced using the name of the cartridge which will instantiate
it (CD account balance). -->
<object RUNAT="SERVER" ID="AccountManagerCD" Cartridge="AccountCApp/AccountC">

<!-- CD account balance. -->
<% $AccountCD = $AccountManagerCD->open($AccName); %>
CD account balance is $<%= $AccountCD->balance() %><P>

</BODY>
</html>
```

LiveHTML ページで使用するために CORBA オブジェクトの IDL インタフェースを生成する

前述の例を実行できるようにするには、CORBA オブジェクトの IDL インタフェースまたはクライアント・スタブがすでに生成され、LiveHTML ページに埋め込まれたスクリプトがそのインタフェースを使用できるようになっている必要があります。これを行うには、Oracle Application Server に付属している IDL-to-Perl コンパイラを使用します。次に、コンパイラのコマンド・ラインの使用例を示します。IDL インタフェース用に生成される PERL バインディングについての詳細は、第 7 章「Perl スクリプトからの CORBA オブジェクトへのアクセス」を参照してください。

```
prompt>   perlidlc -i -I $ORACLE_HOME/public -I $ORACLE_HOME/ows/apps/eco4j/MyAccount
          $ORACLE_HOME/ows/apps/eco4j/server/MyAccount/OASInterfaces.idl
```

MyAccount はユーザーのアプリケーションの名前で、**OASInterfaces.idl** は、ECO/Java オブジェクト用の IDL ソースが入っているファイルです。

JCORBA オブジェクト以外のオブジェクトの場合、コマンド・ラインは次のようになります。

```
prompt>   perlidlc -i -I <include_directory> filename.idl
```

<include_directory> には、IDL ソース **filename.idl** で指定されたインクルード・ファイルが含まれます。

スクリプトの例

次の例は、簡単な埋込みスクリプトの使用方法を示しています。さらに詳しい例は、後の章で各機能の説明とともに紹介します。

Perl のバージョン番号の取得

次のスクリプトは `$]` 変数を使用して、LiveHTML カートリッジが使用する Perl のバージョンを取得します。

```
<p>The LiveHTML cartridge uses Perl version
<%= $] %>
```

インクルードされた Perl モジュールの関数の実行

次のスクリプトは、`Time::localtime` モジュール内の `ctime` 関数を実行します。

```
<p>The local time is:
<% use Time::localtime; %>
<%= ctime(time()) %>
```

5

Web Application Object を使用した開発

Oracle Application Server には、LiveHTML スクリプトからアクセス可能な、多様で拡張可能なオブジェクトのセットが用意されています。このオブジェクトのセットは、トランザクション型 Web ベース・アプリケーションを構築するために必要なランタイム・サービスを提供し、基本的なフレームワークを形成しています。これらのオブジェクトにより、スクリプトから LiveHTML ページの実行または LiveHTML ページの実行時環境の問合せや変更を行うことができます。これらのオブジェクトを使用すると、スクリプトから HTTP ヘッダー情報の取得、Cookie の検索と設定、CORBA 準拠オブジェクトの作成またはアクセス、他のカートリッジへのリクエストの発行、トランザクションのコミットまたはロールバックなどのタスクを実行できます。（トランザクション型の Web Application Object の詳細は、第 6 章「LiveHTML のトランザクション」を参照してください。）

Web Application Object を使用して、複雑なカスタム・ロジックを外部の再利用可能なコンポーネント（ECO/Java オブジェクト、Enterprise Java Beans オブジェクト、JWeb カートリッジ、PL/SQL カートリッジなど）として開発することにより、ユーザー・アプリケーションの機能を拡張できます。これらのコンポーネントは、LiveHTML スクリプトから実行することにより、Web アプリケーション内で使用可能です。

内容

Web Application Object とは

Web Application Object は、Web アプリケーションの実行時環境との対話処理を行うためのオブジェクトのセットを提供しています。これらのオブジェクトは、IDL インタフェースを使用した CORBA オブジェクトとしてインプリメントされており、下位レベルの操作はカプセル化されています。これにより、開発者は Oracle Application Server の拡張機能を使用したコンテンツの開発に専念できます。

Web Application Object の使用例として、HTTP ヘッダーの値の取出し、Cookie の設定、HTML ページのトランザクションの属性の制御、他のカートリッジのオブジェクトおよびメソッドへのアクセスなどの操作があります。

次の表に、現在使用可能な Web Application Object のサマリーを示します。

表 5-1 Web Application Object の概要

オブジェクトのセット	オブジェクト型	説明
コア	Request	ユーザーからの HTTP リクエスト。
	Response	ユーザーのリクエストに対するサーバーのレスポンス。
	HTTPListener	リクエストを受信した Web リスナー。
	Server	オブジェクトの作成とオブジェクト・ファクトリの管理に使用します。
	Document	そのオブジェクトを参照している LiveHTML ページを示すオブジェクト。
	ObjectFactory	Oracle Application Server 環境でオブジェクトの作成または取出しを行います。
コレクション / コンテナ	Vector	動的にサイズ変更が可能な配列ユーティリティ。
	Iterator	コンテナのエレメント間を移動するオブジェクト。
	Hashtable	名前と値のペアをすばやく取り出すユーティリティ。
I/O	HTTPInputStream	クライアントから送信されたデータを読み込むためのストリーム。
	HTTPOutputStream	クライアントへデータを書き込むためのストリーム。

表 5-1 Web Application Object の概要（続き）

オブジェクトのセット	オブジェクト型	説明
ICX（Inter-Cartridge Exchange）	ICXRequest	Oracle Application Server でカートリッジ間のリクエストを作成します。URL を使用してターゲットのカートリッジのアドレスを指定します。このオブジェクトは、トランザクション型のコンテキストの波及をサポートしています。
	ICXResponse	他のカートリッジからデータを取り出します。このオブジェクトは、トランザクション型のコンテキストの波及をサポートしています。
ユーティリティ	Cookie	Cookie の設定と取得のために使用します。
トランザクション	TxContent	トランザクションをコミットまたはロールバックしたり、トランザクションの状態の情報を取得できます。

これらのオブジェクトには、そのオブジェクトの特性を定義するために使用できるメソッドおよび属性が存在します。これらのオブジェクトを使用するには、メソッドを実行して、その属性を読み込むかまたは変更する必要があります。

Web Application Object によるスクリプト作成

Web Application Object には、Web アプリケーションの開発に使用できる属性とメソッドがあります。オブジェクトには、LiveHTML ページに埋め込まれたスクリプトからアクセスします。（詳細は、4-3 ページの「スクリプトの埋込み」の項を参照してください。）オブジェクトおよびそのメソッドと属性を参照する構文は、スクリプト言語の構文に従います。現在、サポートされている言語は Perl なので、このマニュアルの説明も Perl を中心に行います。

Perl の使用

Perl で作成されたスクリプトでは、事前定義済みの次の 5 つの変数を通じて、すべてのオブジェクトのメソッドや属性を参照できます。

```
$Server
$Document
$Request
$Response
$TxContext
```

（詳細は、第 6 章「LiveHTML のトランザクション」を参照）

これらの 5 つの変数は自動的に作成されます。これらを使用する前に宣言や初期化を行う必要はありません。また、Perl には大文字・小文字の区別があります。オブジェクト名の 1 文字目は大文字なので注意してください。

スクリプトの構文では、Web Application Object のメソッドと属性は、該当する変数と矢印演算子（->）を使用して直接実行したりアクセスできます。所有者オブジェクトを指定する必要はありません。たとえば、OutputStream オブジェクトの write() メソッドを実行する場合、次のような構文になります。

```
$Response->write("Hello World¥n");
```

次の表は、Web Application Object を 5 つの Perl 変数の下にグループ化したものです。

$Server	$Request	$Response	$Document	$TxContext
Server ObjectFactory	Request HTTPListener InputStream	Response OutputStream	Document	TxContext （第 6 章「LiveHTML のトランザクション」を参照）

したがって、Request オブジェクトの svr_name 属性を取り出すために、$Request 変数を使用できます。

```
$sname = $Request->svr_name();
```

注意： 属性の中には、読込み専用で、設定不可能なものがあります。

残りの Web Application Object は、コレクション・オブジェクトとユーティリティ /ICX オブジェクトの 2 つのグループに分類できます。

コレクション・オブジェクト	ユーティリティ /ICX オブジェクト
Vector Iterator HashTable	Cookie （パッケージ oracle::OAS::WAO::HTTP::Cookie に存在） ICXRequest （パッケージ oracle::OAS::WAO::OASFrmkObject に存在） ICXResponse （パッケージ oracle::OAS::WAO::OASFrmkObject に存在）

コレクション・オブジェクトは、HashTable（キーの値）と Vector（動的配列）コンテナ・オブジェクト、およびこの 2 つのオブジェクトの操作を行う Iterator オブジェクトで構成されています。これらのオブジェクトはデータの操作に便利です。コンテナ・オブジェクトのメソッドと属性にアクセスするには、どちらかのオブジェクトのリファレンスを含んだ変数を使用する必要があります。たとえば、HTTP ヘッダーの情報を HashTable の中に取り出す場合、Request オブジェクトの get_headers() メソッドを使用して、ヘッダーのリファレンスを取得できます。そして、このリファレンスはヘッダー・データを操作するために Iterator オブジェクトのメソッドによって使用されます。次にスクリプトの例を示します。

```
$headers = $Request->get_headers(); # obtain a reference for the headers
$keylist = $headers->keys(); # obtain a reference for the keys of the header hash
$key = $keylist->get_next_element()->extract();
$value = $headers->get_value($key)->extract(); # value corresponding to key name is
                                                retrieved
```

> **注意：** ヘッダー情報の戻り型は Perl の型にはマップされない IDL Any 型なので、extract() メソッドが必要です。

ユーティリティ・オブジェクトおよび ICX オブジェクトは、Web Application Object のフレームワークに追加の機能を提供しています。それぞれのオブジェクトには固有の関数があります。使用方法の例として、Inter-Cartridge Exchange 操作と Cookie 操作があります。ユーティリティ・オブジェクトと ICX オブジェクトを Perl で使用するには、そのオブジェクトが入っているパッケージ名を指定する必要があります。また、Server オブジェクトの get_object() メソッドを使用して、オブジェクト・リファレンスを取得する必要があります。Cookie オブジェクトにこれらの操作を行うスクリプトの例を次に示します（ICX オブジェクトにも適用できます）。

```
# Load the Perl package containing the Cookie object
use oracle::OAS::WAO::HTTP::Cookie;
$cookie = oracle::OAS::WAO::HTTP::Cookie->narrow($Server->get_object("wao://Cookie"));
# the narrow method is used to narrow the generic CORBA object returned by
# Server.get_object to the correct interface
```

次の項で、Web Application Object を使用したスクリプトの作成例を紹介します。

例

この例は Perl 言語で作成されています。これらのスクリプトを使用するには、スクリプトを <% ...%> タグで囲む必要があります。

Response オブジェクトを使用してクライアントにデータを返す

次の例では、Response オブジェクトの sout 属性を使用して出力ストリーム・オブジェクトを取得してから、出力ストリームの write_wstring() メソッドをコールして、文字列データをクライアントに返します。

```
<%
for ($i = 1; $i <= 5; $i += 2) {
  $Response->sout->write_wstring("The value of i is $i ¥n");
}
%>
```

このスクリプトの出力は次のとおりです。

```
The value of i is 1
The value of i is 3
The value of i is 5
```

Response オブジェクトの write() メソッドを使用しても結果は同じです。次のとおりです。

```
$Response->write("The value of i is $i ¥n");
```

後者の構文は、前者の構文より短い構文を使用して同じ処理を実行できるため、こちらの方をお薦めします。

HTTP ヘッダーを表示する

Request オブジェクトを使用して、リクエスト内の HTTP ヘッダーを表示できます。これを行うスクリプトは次のようになります。

```
<%
# Load the package for HTTP request
use oracle::OAS::WAO::HTTP::HTTPRequest;

$headerhash = $Request->get_headers();   # headerhash is a hashtable
# cycle through the contents of the headerhash hashtable
$keylist = $headerhash->keys();          # keylist is an Iterator
while ($keylist->has_more_elements()) {
    $a_key = $keylist->get_next_element();    # get the next key
    $val = $headerhash->get_value($a_key);    # get the value for the key
    $Response->write("$a_key = $val ¥n");
}
%>
```

この例は、Request、Iterator、Hashtable オブジェクトの使用方法を示すことを目的としています。

仮想パスから物理パスを取得する

次の例は、仮想パスから対応する物理パスを取得します。

```
<%
$listener = $Request->listener();
$ppath = $listener->map_path("/testpages/index.html");
%>
```

Cookie を設定する

Response オブジェクトを使用して、Cookie をクライアントに送信できます。次に例を示します。

```
<%
# Load the package for Cookie
use oracle::OAS::WAO::HTTP::Cookie;

# Utilize the predefined variable $Server to instantiate a
# cookie object.  Since $Server->get_object returns a generic
# CORBA object, you need to narrow it to the right interface,
# i.e. oracle::OAS::WAO::HTTP::Cookie. That will give you an empty cookie.
my $cookie = oracle::OAS::WAO::HTTP::Cookie->narrow($Server->
                                                    get_object("wao://Cookie"));

# set the various attributes of the cookie like name, value, domain,...
$cookie->name("user");
$cookie->value("john");
$cookie->path("");
$cookie->domain("www.foo.com");
$cookie->expires("31-DEC-99 GMT");

# Finally, set the cookie in the "Response" object
$value = $Response->set_cookie($cookie);
%>
```

Cookie を取り出す

```
<%
# Load the package for Cookie and create an object reference to Cookie
use oracle::OAS::WAO::HTTP::Cookie;
my $cookie = oracle::OAS::WAO::HTTP::Cookie->narrow($Server->
                                                    get_object("wao://Cookie"));

# Retrieve our cookie "user"
my $cookie = $Request->get_cookie("user");
use Oracle::hlpr::Excp;
# If there is no exception raised when retrieving the cookie, implying cookie
# is set in client's browser, the cookie value is valid.
if (!Oracle::hlpr::Excp->isexcp($cookie)) {
    $cookie = $cookie->extract();
}
%>
```

PL/SQL プロシージャを実行する

```
<%
# Load the package for ICXRequest and create an object reference to ICXRequest
use oracle::OAS::WAO::OASFrmkObject;
my $icx_object = oracle::OAS::WAO::OASFrmkObject->narrow($Server->
                                                  get_object("wao://ICXRequest"));

# initialize the object with the URL address of the target
$icx_object->init = ("http://machine/plsqlapp/cartx/procedure_name");

# set the Http request method to GET
$icx_object->set_method(GET);

# create a new ICX connection
$resp_object = $icx_object->connect();

# grab the result of the procedure
$foo = $resp_object->content();
...
%>
```

Oracle データベースのデータの問合せと取出し

```
<!-- The following script logs on to an Oracle database, queries for data and
retrieves them; the Perl DBD::Oracle driver is used with the DBI API. HTML
formatting tags are included for completeness -->

<HTML>
<TITLE>OAS LiveHTML Scripting Example</TITLE>
<BODY BGCOLOR="#FFFFFF">
<%
# this subroutine sets up the DB connection
sub set_connection {
    my ($cs, $name, $pwd)=@_;
    use DBI;
    $logon = DBI->connect($cs, $name, $pwd,"Oracle") ||
                          $Response->write("Failed to logon: $DBI::errstr¥n");
}

# this subroutine returns the values of a query
sub get_users {
    my($sqlstmt) = @_;
    $query=$logon->prepare("select username from all_users") ||
                            $Response->write("Failed to perform query
                                              $DBI::errstr¥n");
    $query->execute() ||
            $Response->write("Failed to perform query $DBI::errstr¥n");
```

```
    my $i=0;
    my @emp;
    while(@fields = $query->fetchrow()){
        # need to add the row to an array
        @emp->[$i]=@fields[0];
        $i++;
    }
    return @emp;
}

sub get_user_details {
    my ($username) = @_;
    $query=$logon->prepare("select * from all_users
                           where username = '$username'") ||
                           $Response->write("Failed to perform query
                                            $DBI::errstr¥n");
    $query->execute() ||
    $Response->write("Failed to perform query $DBI::errstr¥n");
    my @fields=$query->fetchrow();
    return @fields;
}
%>

<H2>Server Side Scripting Demonstration</H2>
<P>
<HR ALIGN="LEFT" WIDTH="75%" SIZE="5">
</P>

<%
# Main

# determine if form data was sent
my $formvalue=$Request->form();
if($formvalue){
    my $username=$formvalue->get_value("username")->extract();
    my @details=get_user_details($username);
%>
    <!-- display user details -->
    <!-- outer table -->
    <TABLE BORDER="2" CELLPADDING="10" CELLSPACING="0" WIDTH="75%">
    <TR>
    <TD WIDTH="33%" BGCOLOR="#FFFFCC">USERNAME</TD>
    <TD WIDTH="33%" BGCOLOR="#FFFFCC">USER_ID</TD>
    <TD WIDTH="33%" BGCOLOR="#FFFFCC">CREATED</TD>
    </TR>
    <TR>
    <%
```

```
    my $i=0;
    for(@details){
    %>
    <TD WIDTH="33%" BGCOLOR="#66CCCC"><%=@details[$i++]%></TD>
    </TR>
    </TABLE>

<% } else {%>

    <TABLE BORDER="2" CELLPADDING="10" CELLSPACING="0" WIDTH="75%" HEIGHT="198">
    <TR>
    <TD WIDTH="100%" BGCOLOR="#FFFFCC">Please select a User from below</TD>
    </TR>
    <TR>
    <TD WIDTH="100%">
    <P>
    <!-- inner table -->
    <FORM ACTION="topics.hsp">
    <INPUT TYPE="HIDDEN" NAME="GETDETAILS" VALUE="TRUE">
    <TABLE BORDER="0" WIDTH="100%" CELLSPACING="5" CELLPADDING="5">
    <!-- add each of the users here -->
    <%
    set_connection("machine_name","scott","tiger");
    my @users=get_users();
    my $i=0;

    for(@users){
    %>
        <TR>
        <TD WIDTH="18%" BGCOLOR="#66CCCC">
        <input type="radio" name="username" value="<%=@users[$i]%>"></TD>
        <TD WIDTH="82%" BGCOLOR="#66CCCC"><%=@users[$i++]%></TD>
        </TR>
    <% } %> <!-- for loop -->
    <TR>
    <!-- the padding for the buttons -->
    <TD WIDTH="18%"> </TD>
    <TD WIDTH="82%"> </TD>
    </TR>
    <TR>
    <!-- the form submit buttons -->
    <TD ALIGN="RIGHT" WIDTH="18%"> </TD>
    <TD ALIGN="RIGHT" WIDTH="82%">
    <INPUT TYPE="SUBMIT" VALUE="Show User Details">
    <INPUT TYPE="RESET">
    </TD>
    </TR>
```

```
    </TABLE> <!-- inner-->
    </FORM>
    </P>
    </TD>
    </TR>
    </TABLE> <!-- outer -->
<% } %> <!-- if statement -->
</BODY>
</HTML>
```

メソッドと属性のサマリー

次の表に、各 Web Application Object で使用可能な属性とメソッドのリストを示します。

表 5-2 メソッドと属性のリスト

オブジェクト	メソッド	属性
Request	get_header() get_headers() get_cookie() get_cookies() get_request_info() get_cgivar() init() destroy()	id protocol protocol_major_ver protocol_minor_ver svr_name svr_port form query_string_vars cgivars sin content_len content_type listener auth_type method

表 5-2 メソッドと属性のリスト（続き）

オブジェクト	メソッド	属性
Response	set_header() set_headers() set_cookie() set_cookies() end() redirect() set_status() set_status_and_msg() set_expire_length() set_expire_date() set_content_len() set_content_type() send_headers() write() init() destroy()	id sout keep_alive auto_send headers_sent
HTTPListener	map_path()	host_name port
Cookie		name value path domain expire secure
OutputStream	close() flush() write() write_wstring() write_wchars()	
InputStream	ready() close() read() read_wchars() mark() reset() skip()	mark_supported

表 5-2 メソッドと属性のリスト（続き）

オブジェクト	メソッド	属性
Vector	expand() add_element() add_elements() get_element() get_elements() set_element() set_elements() insert_element() insert_elements() remove_element() remove_elements() elements() lock() unlock() init() destroy()	capacity num_elements capacity_increment auto_resize
Iterator	has_more_elements() get_next_element() get_next_elements() skip() reset() clone_iterator() init() destroy()	
Hashtable	clear() aggregate() get_value() get_values() set_value() set_values() remove_value() remove_values() keys() lock() unlock() init() destroy()	num_keys

表 5-2 メソッドと属性のリスト（続き）

オブジェクト	メソッド	属性
ICXRequest	set_method() set_header() set_headers() set_content() set_contents() set_auth_info() connect() enable_transaction() disable_transaction() init() destroy()	
ICXResponse	get_header() get_headers() init() destroy()	status_code realm reason_phrase http_version using_proxy content
ObjectFactory	get_object()	
Server	get_object_factory() set_object_factory() get_object() get_object_by_type() init() destroy()	
Document		tx_attr dft_script_language

Request

Request オブジェクトのインスタンスは、クライアントが LiveHTML カートリッジにリクエストを送信するときに作成されます。このオブジェクトは、クライアントのリクエスト情報（たとえば、HTTP ヘッダーまたは問合せ文字列情報など）の取出しに使用できます。

表 5-3 Request のメソッド

メソッド	構文と説明

get_header()	`wstring get_header(in wstring header_name)`
	`raises (oracle::OAS::Util::NoSuchElement)`
	指定された HTTP ヘッダー *header_name* の値を返します。

get_headers()	`oracle::OAS::Util::Hashtable get_headers()` すべての HTTP ヘッダーを Hashtable として返します。

get_cookie()	`any get_cookie(in wstring cookie_name)`
	`raises (oracle::OAS::Util::NoSuchElement)`
	指定された Cookie の *cookie_name* の値を返します。

get_cookies()	`oracle::OAS::Util::Hashtable get_cookies()` クライアントが送信した Cookie をすべて返します。
get_request_info()	`wstring get_request_info(in HTTPRequestInfoType type)` リクエストに対応する情報を返します。*type* は、次のいずれかです。 `uri` - リクエストの URI `url` - リクエストの URL `listener_type` - リスナーのタイプとバージョン `virtual_path` - リクエストの仮想パス `physical_path` - リクエストの物理パス `query_string` - 問合せ文字列 `language` - カンマ区切りの言語のリスト `encoding` - カンマ区切りのエンコーディングのリスト `mime_type` - MIME タイプ `user_id` - ユーザー ID `password` - パスワード `ip_addr` - a.b.c.d 表記による IP アドレス

表 5-3 Request のメソッド（続き）

メソッド	構文と説明
get_cgivar()	`wstring get_cgivar(in wstring name)` 指定された CGI 環境変数の値を返します。
init()	`void init(in any init_param)` このオブジェクトを初期化します。*init_param* はオプションです。
destroy()	`void destroy()` このオブジェクトを破棄します。

表 5-4 Request の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
id	R	long	Request オブジェクトのこのインスタンスの識別子。
protocol	R	string	リクエストのプロトコル。
protocol_major_ver	R	long	リクエストのプロトコルのメジャー・バージョン。
protocol_minor_ver	R	long	リクエスト・プロトコルのマイナー・バージョン。
svr_name	R	wstring	このリクエストを受信したサーバーの名前。
svr_port	R	long	このリクエストを受信したポート番号。
form	R	oracle::OAS::Util::Hashtable	Hashtable で返される HTTP リクエストの BODY 内のフォーム・エレメントの値。
query_string_vars	R	oracle::OAS::Util::Hashtable	Hashtable で返される HTTP 問合せ文字列の変数の値。
cgivars	R	oracle::OAS::Util::Hashtable	リクエストに関連付けられた CGI 1.1 変数を取り出し、Hashtable で返します。
sin	R	oracle::OAS::IO::InputStream	クライアントからのリクエスト・エンティティ・データを表す InputStream オブジェクトのインスタンス。
content_len	R	long	リクエスト・エンティティ・データのサイズ（バイト）。

表 5-4 Request の属性（続き）

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
content_type	R	string	リクエスト・エンティティ・データの MIME タイプ。
listener	R	HTTPListener	このリクエストが関連付けられる HTTP リスナー。
auth_type	R	string	リクエストの認証方式。標準方式（基本、Oracle など）のいずれか、またはカスタム方式が使用可能です。
method	R	string	リクエストの方式。使用可能な値は "HEAD"、"GET" または "POST" です。

Response

LiveHTML カートリッジは、Response オブジェクトのインスタンスを作成し、そのリクエストへのレスポンスとしてクライアントに送信します。このオブジェクトは、クライアントへのデータ送信に使用できます。

表 5-5 Response のメソッド

メソッド	構文と説明
set_header()	void set_header(in string *name*, in string *value*) 指定された *name* と *value* のペアを使用して新しい HTTP ヘッダーを追加します。
set_headers()	void set_headers(in oracle::OAS::Util::Hashtable *headers*) Hashtable に含まれる HTTP ヘッダーを追加します。
set_status()	void set_status(in long *status_code*) 返すステータス・コードを設定します。
set_status_and_msg()	void set_status_and_msg(in long *status_code*, in wstring *status_msg*) 返すステータス・コードとステータス・メッセージを設定します。
set_expire_length()	void set_expire_length(in long *expire_length*) ブラウザのキャッシュに入っているページが期限切れになるまでの期間を指定します。
set_expire_date()	void set_expire_date(in oracle::OAS::Util::Date *expire_date*) ブラウザのキャッシュに入っているページが期限切れになる日付と時間を指定します。
set_content_len()	void set_content_len(in long *content_len*) レスポンスのコンテンツの長さを設定します。
set_content_type()	void set_content_type(in wstring *content_type*) レスポンスのコンテンツ・タイプを設定します。
set_cookie()	void set_cookie(in oracle::OAS::WAO::HTTP::Cookie *cookie*) クライアントに返送する Cookie を設定します。
set_cookies()	void set_cookies(in oracle::OAS::Util::Hashtable *cookies*) クライアントに返送する Cookie を設定します。
send_headers()	void send_headers() これまでに設定したヘッダーをすべてクライアントに送信します。
write()	void write(in wstring *msg*) 文字列をクライアントに送信します。

表 5-5 Response のメソッド（続き）

メソッド	構文と説明
end()	`void end()` 処理を停止して結果をクライアントに返します。このメソッドは、OutputStream の `flush()` をコールします。
redirect()	`void redirect(in string new_url)` ブラウザにリダイレクト・メッセージを送信します。その結果、ブラウザは指定された URL への接続を試みます。
init()	`void init(in any init_param)` このオブジェクトを初期化します。*init_param* はオプションです。
destroy()	`void destroy()` このオブジェクトを破棄します。

表 5-6 Response の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
id	R	long	オブジェクトのこのインスタンスの識別子。
sout	R	oracle::OAS::IO::OutputStream	レスポンスの出力ストリーム。
auto_send	R/W	boolean	`true` の場合、レスポンス・ヘッダーを変更する各コールの後でヘッダーが送信されます。
headers_sent	R	boolean	`true` の場合、ヘッダーはすでに送信済みです。
keep_alive	R/W	boolean	HTTP 1.1 KeepAlive が使用可能な場合、値は True です。

Cookie

Cookie オブジェクトは次の表の属性を使用してインプリメントされています。このオブジェクトはクライアントの Cookie について、Netscape 2.0 の仕様に準拠しています。このオブジェクトを使用してクライアントのブラウザに状態情報を保管し、これをその後の HTTP リクエストで再利用できます。

表 5-7 Cookie の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
name	R/W	wstring	この Cookie の名前。
value	R/W	wstring	この Cookie の値。
path	R/W	wstring	この Cookie を提示する URL。URL がこのパスで始まっていない場合、Cookie は提示されません。
domain	R/W	wstring	この Cookie のドメイン。
expires	R/W	wstring	この Cookie の有効期限。この値が指定されないと、クライアントがセッションを終了するときに Cookie は廃棄されます。
secure	R/W	boolean	セキュリティ・フラグの値。デフォルトは false です。

HTTPListener

このオブジェクトは、Oracle Application Server によって管理される Web リスナーを表します。ポート番号など、リスナーに関する情報の取出しに使用できます。

表 5-8 HTTPListener のメソッド

メソッド	構文と説明
map_path()	`string map_path(in string vpath)` 指定された仮想パスに対応する物理パスを返します。

表 5-9 HTTPListener の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
host_name	R	string	その Web リスナーが稼動しているマシンの名前。
port	R	long	その Web リスナーがリスニングするポート番号。

OutputStream

出力ストリームを使用して、データをクライアントに送信できます。通常、Response オブジェクトの sout 属性から出力ストリーム・オブジェクトにアクセスします。たとえば、次のようになります。

```
$Response->sout->write_wstring("Send this line back to the client¥n");
```

この文は、次の文と同じ結果を生成します。

```
$Response->write("Send this line back to the client¥n");
```

注意： ストリームでのマルチバイト文字の使用はサポートされていません。

表 5-10 OutputStream のメソッド

メソッド	構文と説明
close()	`void close()` 出力ストリームをクローズします。
flush()	`void flush()` 書込み先のバッファの内容をフラッシュします。
write()	`void write(in wchar c)` 出力ストリームに 1 文字を書き込みます。これは、文字列を処理する Response オブジェクトの write() メソッドとは異なります。
write_wstring()	`void write_wstring(in wstring s)` 出力ストリームに 1 つ以上の文字を書き込みます。このメソッドは Response オブジェクトの write() メソッドに似ています。
write_wchars()	`void write_wchars(in wcharSeq cSeq)` 出力ストリームに文字配列を書き込みます。

InputStream

入力ストリームを使用して、クライアントからのデータを読み込むことができます。Request オブジェクトの sin 属性から、入力ストリームのインスタンスを取得できます。

注意： ストリームでのマルチバイト文字の使用はサポートされていません。

表 5-11 InputStream のメソッド

メソッド	構文と説明
ready()	`boolean ready()` アプリケーションの入力ストリームが準備できているかどうか。各アプリケーションごとに 1 つだけ、入力ストリームが存在します。
close()	`void close()` 入力ストリームをクローズします。
read()	`wchar read()` 入力ストリームから 1 文字を取得します。
read_wchars()	`long read_wchars(inout wcharSeq` *`cSeq`*`)` 入力ストリームから文字配列を取得します。
mark()	`void mark(in long` *`readLimit`*`)` 指定された位置をマークします。
reset()	`void reset()` 現在位置をマーク済みの位置に移動します。
skip()	`long slip(in long` *`nChars`*`)` 指定された数の文字をスキップし、それらを廃棄します。戻り値はスキップされた文字数です。

表 5-12 InputStream の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
mark_supported	R	boolean	入力ストリームがマークをサポートするかどうか。マークにはブックマークのような機能があります。ストリーム内にマークを置き、後でいつでもそこに戻ることができます。

Vector

Vector は、動的にサイズを変更できる配列として機能します。Vector の最初の索引は 0 です。

表 5-13 Vector の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
capacity	R/W	long	現在の Vector のサイズ。サイズを小さくすると、Vector 内の既存のエレメントがなくなる可能性があります。
num_elements	R/W	long	Vector 内のエレメントの数。
capacity_increment	R/W	long	Vector の増分量。Vector のサイズより大きいエレメントを追加すると、Vector のサイズが増えます。デフォルト値は -1 で、これにより Vector はサイズ変更の際に最適値を選択できます。
auto_resize	R/W	boolean	True の場合、Vector は自動的に容量を増やして、入りきらなかった索引を収容します。 False（デフォルト）の場合、索引が入りきらなかったことを検出すると、Vector は ArrayOutOfBounds 例外を発行します。

表 5-14 Vector のメソッド

メソッド	構文と説明
expand()	`void expand()` `capacity_increment` で指定された量だけ Vector のサイズを増やします。
add_element()	`void add_element(in any element)` 指定されたエレメントを Vector に追加します。必要であれば、Vector はエレメントに合せて自動的にサイズを変更します。
add_elements()	`void add_elements(in anySeq elements)` 指定されたエレメントを Vector に追加します。必要であれば、Vector はエレメントに合せて自動的にサイズを変更します。
get_element()	`any get_element(in long index)` 指定された索引のエレメントを返します。

表 5-14 Vector のメソッド（続き）

メソッド	構文と説明
get_elements()	`anySeq get_elements(in long begin_index, in long num_elements)` *begin_index* から開始して *num_elements* の数のエレメントを返します。
set_element()	`void set_element(in long index, in any element)` 指定された索引のエレメントを設定し、索引の内容を上書きします。`auto_resize` が `False` で、索引が Vector の容量を超えた場合、ArrayOutOfBounds 例外が発生するので注意してください。
set_elements()	`void set_elements(in long begin_index, in anySeq elements)` *begin_index* からエレメントの設定を開始します。これらの索引のエレメントはすべて上書きされます。`auto_resize` が `False` で、索引が増加して Vector の容量を超えた場合、ArrayOutOfBounds 例外が発生するので注意してください。
insert_element()	`void insert_element(in long index, in any element)` 指定された索引にエレメントを挿入し、索引の以前のエレメントとその上位にあるエレメントをすべて 1 ポジション上に移動します。`auto_resize` が `False` で、索引が Vector の容量を超えた場合、ArrayOutOfBounds 例外が発生するので注意してください。
insert_elements()	`void insert_elements(in long begin_index, in anySeq elements)` *begin_index* からエレメントの挿入を開始し、以前にそのポジションにあったエレメントは上に移動します。`auto_resize` が `False` で、索引が Vector の容量を超えた場合、ArrayOutOfBounds 例外が発生するので注意してください。
remove_element()	`void remove_element(in long index)` 指定された索引のエレメントを削除し、指定された索引よりも大きい索引のエレメントは 1 つ下に移動します。エレメントの削除は、エレメントを単に Vector から削除するだけなので注意してください。エレメントそのものを破棄するわけではありません。指定された索引にエレメントがない場合、NoSuchElement 例外が発生します。
remove_elements()	`void remove_elements(in long begin_index, in long num_elements)` *begin_index* から開始して *num_elements* 個のエレメントを削除します。この索引（begin_index + num_elements）よりも大きい索引のエレメントはすべて下に移動されます。
elements()	`Iterator elements()` 新規の Iterator を作成します。これは、Vector を通過（トラバース）してその内容を取得するために使用します。Iterator を確実に有効にするために、Iterator を作成する前に Vector をロックして、Iterator を破棄した後でロックを解除します。Vector がロックされている間に Vector を変更する必要がないようにしてください。
lock()	`void lock()` Vector をロックして、内容を変更できないようにします。

表 5-14 Vector のメソッド（続き）

メソッド	構文と説明
unlock()	`void unlock()` Vector のロックを解除します。
init()	`void init(in any init_param)` このオブジェクトを初期化します。`init_param` はオプションです。
destroy()	`void destroy()` このオブジェクトを破棄します。

Iterator

コンテナ内のエレメント間を移動するオブジェクトです。一連のエレメントごとに、このオブジェクトのインスタンスが 1 つ作成されます。インスタンスが作成されると、一連のエレメントの先頭のエレメントの前に位置づけられます。

表 5-15 Iterator のメソッド

メソッド	構文と説明
has_more_elements()	`boolean has_more_elements()` 一連のエレメントにまだエレメントが含まれている場合は、True を返します。
get_next_element()	`any get_next_element()` 次のエレメントを返し、Iterator をその後のエレメントに移動します。
get_next_elements()	`anySeq get_next_elements(in long n)` 次の n 個のエレメントを返し、Iterator を n ポジションだけ進めます。
skip()	`void skip(in long n)` 次の n 個のエレメントをスキップします。
reset()	`void reset()` Iterator の位置を先頭のエレメントに戻します（存在する場合）。
clone_iterator()	`Iterator clone_iterator()` 新規の Iterator のインスタンスを既存のものと同じ場所に作成します。
init()	`void init(in any init_param)` このオブジェクトを初期化します。`init_param` はオプションです。
destroy()	`void destroy()` このオブジェクトを破棄します。

Hashtable

Hashtable は、名前と値のペアの格納と取出しをすばやく行えるようにするメカニズムです。

表 5-16 Hashtable の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
num_keys	R	long	Hashtable 内のキーの数。

表 5-17 Hashtable のメソッド

メソッド	構文と説明
clear()	`void clear()` Hashtable からすべてのキーを削除します。
aggregate()	`void aggregate(in Hashtable table)` 指定された Hashtable の内容をこの Hashtable に追加します。
get_value()	`any get_value(in wstring key)` 指定されたキーに関連付けられた値を返します。
get_values()	`anySeq get_values(in wstringSeq keys)` 指定されたキーに関連付けられた複数の値を返します。
set_value()	`void set_value(in wstring key, in any value)` キーと値のペアを Hashtable に追加します。
set_values()	`void set_values(in wstringSeq keys, in anySeq values)` 複数のキーと値のペアを Hashtable に追加します。
remove_value()	`void remove_value(in wstring key)` キーとその値を Hashtable から削除します。
remove_values()	`void remove_values(in wstringSeq keys)` 指定された複数のキーとその値を Hashtable から削除します。

表 5-17 Hashtable のメソッド（続き）

メソッド	構文と説明
keys()	`Iterator keys()` Hashtable のキーを取得するために通過（トラバース）する Iterator を返します。Iterator を確実に有効にするために、Iterator を作成する前に Hashtable をロックして、Iterator を破棄した後でロックを解除します。Hashtable がロックされている間に Hashtable を変更する必要がないようにしてください。
lock()	`void lock()` Hashtable をロックして、変更できないようにします。
unlock()	`void unlock()` Hashtable のロックを解除します。
init()	`void init(in any init_param)` このオブジェクトを初期化します。`init_param` はオプションです。
destroy()	`void destroy()` このオブジェクトを破棄します。

ICXRequest

ICXRequest オブジェクトは、LiveHTML カートリッジから Oracle Application Server の他のカートリッジへのリクエストを作成するために使用されます。

表 5-18 ICXRequest のメソッド

メソッド	構文と説明
set_method()	`void set_method(in HTTPRequestMethod method)` このリクエストの方式を設定します。設定できる方式は、HTTP 1.1 仕様で指定されている GET、HEAD、POST、PUT、DELETE、TRACE のいずれかです。
set_header()	`void set_header(in wstring name, in wstring value)` このリクエストのシングル・ヘッダーを設定します。`name` はヘッダーの名前、`value` はヘッダーの値です。
set_headers()	`void set_headers(in oracle::OAS::Util::Hashtable headers)` 指定されたヘッダーを既存のヘッダーに追加します。`headers` は、既存の一連のヘッダーが入っている Hashtable で、この後ろにヘッダーが付加されます。Hashtable の内容は wstring、Cookie、Vector 型のいずれかなので注意してください。Vector 型の場合、Vector の各エントリは文字列または Cookie である必要があります。
set_content()	`void set_content(in wstring name, in wstring value)` このリクエストの内容を設定します。`name` は内容の名前、`value` は内容の値です。
set_contents()	`void set_contents(in oracle::OAS::Util::Hashtable contents)` Hashtable の内容をリクエストの既存のコンテンツに追加します。`contents` は新しい項目が含まれる Hashtable です。
set_auth_info()	`void set_auth_info(in wstring name, in wstring password)` このリクエストのユーザー名とパスワードを設定します。
connect()	`ICXResponse connect()` ICX 接続を確立し、新規 ICX 接続を示す ICXResponse オブジェクトを返します。
enable_transaction()	`void enable_transaction()` トランザクションのコンテキストを現行の ICX リクエストを使用して伝播させます。
disable_transaction()	`void disable_transaction()` 現行の ICX リクエストを使用したトランザクション・コンテキストの伝播を使用禁止にします。

表 5-18 ICXRequest のメソッド（続き）

メソッド	構文と説明
init()	`void init(in wstring url)` ターゲット・オブジェクトの場所を設定します。`url` は URL アドレスです。
destroy()	`void destroy()` このオブジェクトを破棄します。

ICXResponse

ICXResponse は、ICXRequest に応答する他のカートリッジからの返信を取得するために使用されるオブジェクトです。

表 5-19 ICXResponse のメソッド

メソッド	構文と説明
get_header()	`wstring get_header(in wstring header_name)` 指定された `header_name` に関連付けられたヘッダー値を取得します。
get_headers()	`oracle::OAS::Util::Hashtable get_headers()` Hashtable 内の ICXResponse オブジェクトに関連付けられたすべてのヘッダーの値を取得します。
init()	`void init(in any init_param)` このオブジェクトを初期化します。`init_param` はオプションです。
destroy()	`void destroy()` このオブジェクトを破棄します。

表 5-20 ICXResponse の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
status_code	R	long	HTTP レスポンス・コードを示します。
realm	R	wstring	レスポンスに指定された認証レルムの名前を示します。
reason_phrase	R	wstring	HTTP レスポンス・コードに対する理由の文字列を示します。
http_version	R	wstring	使用中の HTTP のバージョン。
using_proxy	R	boolean	リクエストにプロキシが使用されたかどうかを示します。
content	R	oracle::OAS::IO::InputStream	内容を入力ストリーム（テキスト・ベース）として読み込みます。

ObjectFactory

このオブジェクトは、新規オブジェクトを作成したり、既存のオブジェクトを取り出します。

表 5-21 ObjectFactory のメソッド

メソッド	構文と説明
get_object()	`Object get_object(in wstring obj_name)` `obj_name` で指定されたファクトリ固有の名前に従って、既存のオブジェクトを取り出すか、または新規のオブジェクトを作成します。 たとえば、Oracle Application Server に用意されているデフォルト・ファクトリの 1 つである WAOObjectFactory では、`obj_name` として Vector、Hashtable、Cookie および ICXRequest の値が有効です。

Server

Server オブジェクトは、Oracle Application Server の実行時環境を表します。これは、オブジェクトの作成機能を提供するために使用されます。

表 5-22 Server のメソッド

メソッド	構文と説明
get_object_factory()	`ObjectFactory get_object_factory(in wstring obj_factory_type)` 指定されたオブジェクト型のオブジェクト・ファクトリを取得します。`obj_factory_type` に有効な値は次のとおりです。 "wao" - Web Application Object のファクトリを表します。 "ior" - ストリング化された CORBA オブジェクト・リファレンスによって指定されたオブジェクトのファクトリを表します。 "cartx" - Oracle Application Server カートリッジ・オブジェクトのファクトリを表します。
get_object()	`Object get_object(in wstring name)` 指定された名前を使用してオブジェクトの取出しまたは作成を行います。要求されたオブジェクトを示すオブジェクト・リファレンスを返します。オブジェクトが見つからない場合または作成できない場合は、NULL 値を返します。`name` の前に次のいずれかの接頭辞を付ける必要があるので注意してください。 "wao://" - WAO Framework Object Set（どの Web Application Object でも可）。 "ior://" - ストリング化された CORBA オブジェクト・リファレンス。 "cartx://" - Oracle Application Server カートリッジの名前。
get_object_by_type()	`Object get_object_by_type (in wstring type, in wstring name)` オブジェクトの型と名前に基づいてオブジェクトの取出しまたは作成を行います。要求されたオブジェクトを示すオブジェクト・リファレンスを返します。オブジェクトが見つからない場合または作成できない場合は、NULL 値を返します。名前は、次のいずれかにする必要があります。 "wao" - WAO Framework Object Set。 "ior" - ストリング化された CORBA オブジェクト・リファレンス。 "cartx" - Oracle Application Server カートリッジの名前。
init()	`void init(in any init_param)` このオブジェクトを初期化します。`init_param` はオプションです。
destroy()	`void destroy()` このオブジェクトを破棄します。

Document

Document オブジェクトは、それがインストールされている各 LiveHTML ページを表します。これを使用すると、各ページの属性にアクセスできます。

表 5-23 ドキュメントの属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
tx_attr	R	string	現在の HTML ページのトランザクションのコンテキストを示す値を返します。 0 = トランザクションに参加する必要がある 1 = 新規トランザクションを開始 2 = トランザクション内 3 = トランザクション外
dft_script_language	R	wstring	現在の HTML ページのデフォルトのスクリプト言語を示す文字列を返します。現在のリリースで有効な値は PERL です。

6

LiveHTML のトランザクション

Web Application Object を使用して、トランザクション機能を持つアプリケーションを開発できます。これらの機能は、トランザクション・プロパティが使用可能になっている各 LiveHTML ページに、ページ単位で組み込まれます。関係する各カートリッジがトランザクションをサポートしている場合、トランザクションは複数のカートリッジにまたがることができます。詳細は、『Oracle Application Server 管理者ガイド』を参照してください。

内容

LiveHTML ページのトランザクションのプロパティの指定

LiveHTML ページのディレクティブ・タグを使用して、その LiveHTML ページがトランザクションを使用可能であると宣言できます。同じタグを使用して、そのページがトランザクションを使用可能な実行スレッド内でコールされても、トランザクションをサポートしないように指定することもできます。このタグの構文は次のとおりです。

```
<%@ Transaction= attribute %>
```

attribute は表 6-1「トランザクションの属性とその機能」の 4 つの属性のいずれかになります。このタグは LiveHTML ページのどこにでも挿入できます。後でこの章で説明する Web Application Object のトランザクション・オブジェクトを使用するためには、このタグが必要です。

表 6-1 トランザクションの属性とその機能

属性	アクション	クライアントの現在のトランザクションの状態	LiveHTML ランタイムのアクション
required	トランザクション・コンテキストが必要であることを示します。	トランザクションではない	トランザクション #2 の開始 スクリプトの実行 トランザクション #2 の終了
		トランザクション #1 内	トランザクション #1 の継承
requires_new	新規トランザクションが必要であることを示します。 ネストされたトランザクションは作成されないので注意してください。	トランザクションではない	トランザクション #2 の開始 スクリプトの実行 トランザクション #2 の終了
		トランザクション #1 内	トランザクション #1 の保留 トランザクション #2 の開始 スクリプトの実行 トランザクション #2 の終了 トランザクション #1 の再開
supported	トランザクションがサポートされていることを示します。	トランザクションではない	トランザクションではない
		トランザクション #1 内	トランザクション #1 の継承
not_supported	トランザクションがサポートされていないことを示します。	トランザクションではない	トランザクションではない
		トランザクション #1 内	トランザクション #1 の保留 スクリプトの実行 トランザクション #1 の再開

たとえば、現在、LiveHTML がトランザクションを処理中で、HTML ページが `<%@ Transaction = not_supported %>` タグによってコールされた場合、LiveHTML はそのタグを検出し、現行のトランザクションを保留にして、その HTML ページのスクリプトを実行します。スクリプトの実行が完了したら、LiveHTML はトランザクションを再開します。

"supported" 属性または "requires_new" 属性が指定されると、LiveHTML は次のアクションを実行します。

1. リクエストが既存のトランザクションに関連付けられているかどうかをチェックします。
2. 必要であれば、新規のトランザクションを開始します（たとえば、`<%@Transaction = requires_new %>`）。
3. その HTML ページの処理の完了時に、トランザクションをコミットします。

Web Application Object のトランザクション・オブジェクト

トランザクションに対応できるように、Web Application Object（WAO）にいくつかのオブジェクトが作成されました。これらのオブジェクトはメソッドおよび属性をインプリメントしています。これらのメソッドおよび属性は定数を返し、アプリケーションのトランザクション処理を助けます。（これらのオブジェクトのいずれかを使用する LiveHTML ページには、この章で前述したように、必ず `<%@ Transaction = attribute %>` タグが指定されている必要があります。）

表 6-2 WAO のトランザクション・オブジェクト

オブジェクト	メソッド	属性	定数
TxContext	commit() rollback()		TX_OUTSIDE TX_ROLLBACK TX_MIXED TX_HAZARD TX_PROTOCOL_ERROR TX_ERROR TX_FAIL TX_EINVAL TX_COMMITTED TX_NO_BEGIN TX_ROLLBACK_NO_BEGIN TX_MIXED_NO_BEGIN TX_HAZARD_NO_BEGIN TX_COMMITTED_NO_BEGIN
TxScriptDoc		tx_attr dft_script_language	

注意： ICXRequest オブジェクトと ICXResponse オブジェクトはトランザクションをサポートしています。詳細は、第 5 章「Web Application Object を使用した開発」を参照してください。

TxContext

このオブジェクトを使用して、トランザクションをコミットまたはロールバックしたり、トランザクションの状態情報を取得できます。

表 6-3 TxContext のメソッド

メソッド	説明
commit()	現行のトランザクションをコミットします。
rollback()	現行のトランザクションをロールバックします。

表 6-4 TxContext の定数（Open Group による XA 仕様に準拠）

属性	説明
TX_OUTSIDE	値 = -1 トランザクションはローカル・トランザクションです。
TX_ROLLBACK	値 = -2 トランザクションはロールバックされました。
TX_MIXED	値 = -3 トランザクションの一部がコミットされ、一部がロールバックされました。
TX_HAZARD	値 = -4 トランザクションの一部がコミットされ、一部がロールバックされた可能性があります。
TX_PROTOCOL_ERROR	値 = -5 不適切なコンテキストの中でルーチンが実行されました。
TX_ERROR	値 = -6 一時的なエラーが発生しました。
TX_FAIL	値 = -7 致命的エラーが発生しました。リクエストは処理されますが、インスタンスは破棄されます。
TX_EINVAL	値 = -8 無効な引数が指定されました。
TX_COMMITTED	値 = -9 トランザクションはヒューリスティックにコミットされました。

表 6-4 TxContext の定数（Open Group による XA 仕様に準拠）（続き）

属性	説明
TX_NO_BEGIN	値 = -100 トランザクションがコミットされましたが、新規トランザクションを開始できませんでした。
TX_ROLLBACK_NO_BEGIN	値 = TX_ROLLBACK + TX_NO_BEGIN トランザクションがロールバックされましたが、新規トランザクションを開始できませんでした。
TX_MIXED_NO_BEGIN	値 = TX_MIXED + TX_NO_BEGIN トランザクションの一部がコミット、一部がロールバックされ、新規トランザクションを開始できませんでした。
TX_HAZARD_NO_BEGIN	値 = TX_HAZARD + TX_NO_BEGIN トランザクションの一部がコミット、一部がロールバックされた可能性があり、新規トランザクションを開始できませんでした。
TX_COMMITTED_NO_BEGIN	値 = TX_COMMITTED + TX_NO_BEGIN トランザクションがヒューリスティックにコミットされ、新規トランザクションを開始できませんでした。

TxScriptDoc

TxScriptDoc オブジェクトは、HTML ページのトランザクション属性を示します。

表 6-5 TxScriptDoc の属性

属性	読込み専用（R）または読込み / 書込み（R/W）	型	説明
tx_attr	R	TxAttr	HTML ページのトランザクション・プロパティを返します。この属性には、次の値のいずれかが入ります。 0 - そのページはトランザクションに参加する必要があります。 1 - そのページが新規トランザクションを開始します。 2 - トランザクション内で実行できます。 3 - トランザクション内では機能しません。
dft_script_language	R/W	lstring	使用されるデフォルトのスクリプト言語を指定します。

例

次の例では、".hsp" ファイルに埋め込まれた Perl スクリプトにより、トランザクションのコンテキストで Web Application Object を使用して、PL/SQL ストアド・プロシージャからの結果を要求しています。

```
...
<%@ Transaction = required %>
...
<%
# Load the package for ICXRequest and create an object reference to ICXRequest
use oracle::OAS::WAO::OASFrmkObject;
my $icx_object = oracle::OAS::WAO::OASFrmkObject->narrow($Server->
                                                    get_object("wao://ICXRequest"));

# initialize the object with the URL address of the target
$icx_object->init = ("http://machine/plsqlapp/cartx/procedure_name");

# set the Http request method to GET
$icx_object->set_method(GET);

# create a new ICX connection
$resp_object = $icx_object->connect();

# grab the result of the procedure
$foo = $resp_object->content();
...
%>
```

7

Perl スクリプトからの CORBA オブジェクトへのアクセス

IDL（Interface Definition Language）は、CORBA オブジェクトのプログラミング・インタフェースを指定するために使用される業界標準の言語です。CORBA（Common Object Request Broker Architecture）は、分散オブジェクト指向プログラミングの業界標準モデルです。どちらの標準も、Object Management Group（OMG）によって定義され、管理されています。CORBA と IDL の詳細は、OMG の Web サイト **http://www.omg.org** を参照してください。

CORBA オブジェクトを開発する場合、オブジェクトの開発者が IDL を使用してオブジェクトのパブリック・インタフェースを定義し、公開します。次に、クライアント・プログラマが、使用可能ないくつかの IDL コンパイラのいずれかを使用して、特定のプログラム言語でオブジェクトのインタフェース・バインディングを生成します。クライアントのプログラマは、その言語で作成されたバインディングを使用して、自分のプログラムからオブジェクトにアクセスします。

Oracle では、IDL-to-Perl コンパイラを提供しています。このコンパイラを使用して CORBA オブジェクト用の Perl バインディングを生成すると、LiveHTML ドキュメントに埋め込まれた Perl スクリプトからオブジェクトにアクセスでできます。この拡張機能により、Live HTML アプリケーションの開発者は、他の CORBA オブジェクトで実現されたサービスを活用できます。また、LiveHTML の開発者にとって、言語およびプラットフォームの選択肢も大幅に広がります。

内容

IDL-to-Perl コンパイラの使用

IDL-to-Perl コンパイラは perlidlc という名前で、Oracle Application Server マシンの次のディレクトリに存在します。

```
$ORAWEB_HOME/bin
```

次のようにして、IDL-to-Perl コンパイラを実行します。

1. `cd $ORAWEB_HOME/bin`
2. `./perlidlc` *`idl-file-path`*

idl-file-path は、Perl バインディングを生成する IDL ファイルのフルパス名です。

IDL-to-Perl コンパイラは、デフォルトで、生成したファイルを $ORAWEB_HOME/../cartx/livehtml/stubs/ の 2 つのサブディレクトリに入れます。

- perl/— このサブディレクトリには、生成された Perl にマップされたパッケージ（.pm ファイル）が含まれます。
- java/— このサブディレクトリには、Perl にマップされたパッケージに必要な生成済みの Java マッピングが含まれます。

> **注意：** perlidlc を使用して Perl バインディングを生成し、<OBJECT> タグによって作成されたオブジェクト ID を通じてこれらのバインディングを使用する方法の例は、第 4 章「スクリプトの作成」の「<OBJECT> タグ」の項を参照してください。

Oracle Application Server 4.0 用 IDL-to-Perl コンパイラの概要

基礎となる Java スタブ（Perl スタブではない）は、ランタイムの引数の範囲チェックを行います。

Perl では、現在、型の検証は行われていないため、IDL 操作に正しくマップされた型が指定されているかどうかを確認してください。型の検証は、次のリリースで追加される予定です。

このリリースでは、静的インタフェースの実行のみサポートされており、動的インタフェースの実行に必要な擬似オブジェクト API はサポートされていません。

その他のコンパイラ・オプション

IDL-to-Perl コンパイラのその他のオプションは次のとおりです。

- `-I` *`include-dir`*

  `include-dir` には、コンパイルされる IDL ファイルに必要なソース・コード・ファイルが入っているディレクトリを指定します。このオプションを複数回繰り返して、複数のディレクトリを指定できます。

- `-D` *`symbol-name`*

  symbol-name には、プリプロセス中に " 定義済み " とみなされるプリプロセッサのシンボルを指定します（IDL ファイルには、プリプロセッサ・ディレクティブを C の標準のプリプロセッサである cpp の形式で指定できます）。コマンド・ラインでこのオプションを指定しても、ソース・コード内でプリプロセッサ・ディレクティブである `#define` を使用しても同じです。このオプションを複数回繰り返して、複数のシンボルを定義できます。

- *`-O output-dir`*

  `-O output dir` コマンドを使用して、生成されたスタブを *`output-dir`*`/perl` と *`output-dir`*`/java` に入れます。**wrb.app** ファイルを設定して、*`output-dir`*`/perl` が LiveHTML カートリッジの PERLLIB 環境変数に、*`output-dir`*`/java` が LiveHTML カートリッジの CLASSPATH 環境変数に含まれるようにしてください。

  Java スタブはシームレスなので、Perl プログラマが Java スタブ・レイヤーを意識することはありません。

注意： *`-O output-dir`* が指定されていない場合、`perlidlc` は、デフォルトで `$ORAWEB_HOME/ows/cartx/livehtml/stubs/`*`{perl|java}`* ディレクトリの下にスタブを生成します。これらのディレクトリは、すでに LiveHTML カートリッジの PERLLIB 環境と CLASSPATH 環境に含まれているため、デフォルト・ディレクトリに生成されたスタブには、LiveHTML ページから簡単にアクセスできます。（NT システムの場合、各環境変数は 512 バイト以下に制限されているので注意してください。CLASSPATH については、このサイズ制限を超えても構いません。）

識別子、ネーミング・スコープおよび Perl パッケージ

IDL-to-Perl コンパイラは、各 IDL の定義を同じ名前の Perl 識別子にマップします（スコープなしにするか、またはパッケージ内をスコープにできます）。

次の種類の IDL 定義は、独自のネーミング・スコープを作成します。

- `module`
- `interface`

- struct
- union
- enum
- exception

> **注意：** 現在、union の Perl バインディングはインプリメントされていません。

たとえば、コンパイラは idlmod という名前の、スコープを持たない IDL モジュールを idlmod という名前の Perl パッケージにマップします。idlmod は、出力ディレクトリの perl/ サブディレクトリ内に新規作成されたファイル idlmod.pm の中に存在します。このモジュールをサポートすることにより、定数 default_width が定義され、Perl クライアントから次の方法でこの定数にアクセスできます。

```
use idlmod;
...
$width = $idlmod::default_width;
```

この例の use ディレクティブについては、「生成された Perl モジュールへのアクセス」で説明します。

ネストされたスコープ

他の IDL スコープの中で宣言されるスコープを作成する IDL 識別子は、スコープを持たない識別子と同様にマップされます。しかし、生成された Perl モジュール・ファイルは親スコープの名前をベースに命名されたサブディレクトリの中に格納され、生成された Perl パッケージは親パッケージ内をスコープとする点が異なります。

たとえば、サブモジュール inner を宣言する IDL モジュール outer の場合、IDL-to-Perl コンパイラは、少なくとも次の 2 つの Perl モジュール・ファイルを出力ディレクトリの perl/ サブディレクトリの中に生成します。

- outer.pm
- outer/inner.pm

モジュール inner で定数 right を宣言した場合、Perl クライアントから次の方法でこの定数にアクセスできます。

```
use outer;
...
$align = $outer::inner::right;
```

生成された Perl モジュールへのアクセス

マップされたパッケージにアクセスするには、use または require ディレクティブの引数として、パッケージ名を指定する必要があります。たとえば、次のようになります。

```
use idlmod;
```

アクセスするマップ済みのパッケージそれぞれについて、use または require ディレクティブを指定する必要があります。

スクリプトで IDL 識別子 <ident> にマップされたパッケージを明示的にインクルードすると、<ident> 内をスコープとするすべての IDL 識別子にマップされているパッケージが自動的にインクルードされます。たとえば、前述の例の use outer; コマンドは outer と outer::inner をインクルードするので、$outer::inner::right; へのアクセスが許可されます。

データ型

Perl はいろいろな種類のスカラー型を区別しません。IDL-to-Perl コンパイラは、最も基本的な IDL のデータ型を Perl のスカラー型にマップします。コンパイル時に Perl スクリプトの型チェックが行われないかわりに、CORBA オブジェクトの操作のコール用に生成された Perl コードが、スカラー型にマップされたパラメータに渡される値について、実行時に範囲の検証を行います。

配列リファレンスやハッシュ・リファレンスなど、他の Perl 型にマップされたパラメータについては、生成された Perl コードが、渡された値が有効な Perl 型にマップされた型であるかどうかをチェックします。

IDL の構造体や共用体など、作成された型（詳細はこの後の「生成された Perl バインディングの使用」を参照）については、生成された Perl コードが作成された型のフィールドの型チェックと範囲検証を繰り返し行います。

生成された Perl バインディングの使用

この項では、CORBA オブジェクトの Perl クライアントが、IDL-to-Perl コンパイラによって生成されたバインディングを使用する方法を示します。次の各項では、IDL の特定の種類の定義の例とそれに対応する Perl コードの例を、クライアントの使用方法とともに説明します。

module

IDL の module はスコープの作成の定義です。IDL-to-Perl コンパイラは、IDL の module を Perl パッケージにマップします。Perl パッケージは、module で宣言されている識別子にスコープを提供します。module の名前は、ローカルの ORB システム内で一意である必要があります。

例

```
// IDL
module finance {
    const long L = 3;

    ...
};
```

Perl クライアントの使用方法

```
# Perl client
use finance;
...
$longval = $finance::L;
```

例 : ネストされた module

IDL-to-Perl コンパイラは、他の IDL module 内で宣言されている IDL module を、親モジュールを実現しているパッケージ内に含まれるパッケージにマップします。

```
// IDL
module outer {
    module inner {
        const short thing = 3;
        ...
    };
    ...
};
```

Perl クライアントの使用方法

```
# Perl client
use outer;
...
$intval = $outer::inner::thing;
# outer::inner package is automatically included while including outer
```

オブジェクト・リファレンス

CORBA オブジェクトへのインタフェースは、IDL では `interface` 定義で定義されます。Perl のクライアント・プログラムから CORBA オブジェクトの操作を実行するには、そのオブジェクトのリファレンスを取得する必要があります。これを実現するために、IDL-to-Perl コンパイラは、CORBA オブジェクトへのインタフェースとして機能するすべてのパッケージ内で、`bind()` クラス（パッケージ）・メソッドを使用可能にします。`bind()` は、対応する CORBA サーバー・オブジェクトのプロキシ・オブジェクトへのリファレンスを作成し、返します。

メソッド名の前にパッケージ名を付けることにより、このオブジェクトと、オブジェクト・リファレンスを持たないその他のすべてのクラス・メソッドを実行します。

例

```
// IDL
module finance {
    interface account {
        ...
    }

    interface bank {
        account newaccount(in string name);
        account getaccount(in string name);
        ...
    }
    ...
}
```

Perl クライアントの使用方法

finance::bank 型のオブジェクト・リファレンスを取得するには、interface bank が CORBA::Object から継承する bind() メソッドを Perl クライアント・コードで使用する必要があります。

```
# Perl client
use finance;
...
```

注意： これらのメソッドをコールするには、パッケージのデリミタ演算子（::）ではなく、矢印演算子（->）を使用する必要があります。Perl5 インタプリタでこれらのメソッドの引数を正しく処理するために、矢印演算子を使用する必要があります。

これで、オブジェクト・リファレンス $bank を使用して account オブジェクトのメソッドを実行できます。

```
# Perl client
...
$account = $bank->getaccount("Joseph P. Shmuck");
```

注意： クライアント・プログラムがいずれかのパッケージの `bind()` メソッドを初めてコールするときに ORB が自動的に初期化され、それ以降のオブジェクトやクライアントからの ORB コールによって使用できるようにします。

絞込み（narrow）

インタフェース `A` をインプリメントしているオブジェクト・リファレンスがあり、実際にはこのリファレンスはインタフェース `B` をインプリメントしているオブジェクトを参照することがわかっているとします。インタフェース B をインプリメントしているオブジェクト・リファレンスを取得することにより、リファレンスを " 絞り込む " ことができます。これを行うには、次のようにして、インタフェース B をマップするクラス class `B` の `narrow()` クラス・メソッドを使用します。

```
$Bref = B->narrow($Aref);
```

これが間違っていて、実際には `$Aref` がインタフェース `B` をインプリメントしているオブジェクトを参照していない場合、このメソッドにより例外が発生します（後述の「exception」を参照）。

interface

IDL-to-Perl コンパイラは、IDL の interface を Perl パッケージにマップします。

例

```
// IDL
module finance {
    interface bank {
        const short stuff = 3;
        ...
    };
    ...
};
```

IDL の interface bank は `finance::bank` パッケージにマップされます。

Perl クライアントの使用方法

```
# Perl client
use finance; #also uses finance::bank
...
$var = $finance::bank::stuff;
```

例 : 継承

IDL の継承のマッピングは、IDL の interface がマップされているパッケージ上で、Perl の継承メカニズムによって実現されます。

```
// IDL
module finance {
    interface account {
        double getbalance();
        ...;
    };
    interface checkingaccount : account {
        ...
    };
    interface bank {
        checkingaccount newchecking(in string name);
        checkingaccount getchecking(in string name);
        ...
    }
};
```

Perl クライアントの使用方法

```
# Perl client
use finance;
...
$bank = finance::bank->bind();
$checking = $bank->getchecking("Joseph P. Shmuck");
$balance = $checking->getbalance();
```

Perl のマップ済みパッケージ `finance::checkingaccount` は `finance::account` から継承されるため、`$checking` 上の `getbalance` メソッドの実行は有効です。これは、`finance::checkingaccount` のインスタンスである `$checking` が `finance::account` から `getbalance` を継承するためです。

定数

IDL-to-Perl コンパイラは、定数が宣言されている IDL スコープに対応するパッケージ内の Perl スカラーに IDL 定数をマップします。

例

```
// IDL
module finance {
    interface account {
        const double minbalance = 500.0;
        ...
    };
    ...
};
```

Perl クライアントの使用方法

```
# Perl client
use finance;
...
$minbal = $finance::account::minbalance;
```

基本データ型

IDL-to-Perl コンパイラは、次の IDL のデータ型を Perl のスカラー型にマップします。

- `short`
- `long`
- `long long`
- `unsigned short`
- `unsigned long`
- `unsigned long long`
- `float`
- `double`
- `char`
- `wchar`
- `octet`
- `boolean`

生成された Perl コードが実行時の型チェックと範囲チェックを実行する方法については、前述の「データ型」を参照してください。

変数やパラメータの値、および IDL 型のブール演算の戻り値は、ブールのコンテキストで評価できる Perl のスカラー値にマップされます。

IDL の Any 型

IDL-to-Perl コンパイラは、`Any` 型の変数またはパラメータを `CORBA::Any` 型の CORBA の擬似オブジェクト・リファレンスにマップします。この擬似オブジェクト・インタフェースの詳細は、第 8 章「Perl クライアントの CORBA 擬似オブジェクト API」を参照してください。

例

```
// IDL
module finance {
    interface account {
        Any collateral(in any asset);
        ...
    };
    interface bank {
        account newaccount(in string name);
        account getaccount(in string name);
        ...
    };
};
```

Perl クライアントの使用方法

```
# Perl Client
use finance;
...
$bank = finance::bank->bind();
$acct = $bank->getaccount("Joseph P. Shmuck");

# create a CORBA::Any pseudo-object and get a reference to it
$asset = CORBA::TypeCode->create_any();

# for some value, construct a TypeCode and store it in $tc--
# suppose for this example it's a short integer type
$value = 12;
$tc = CORBA::TypeCode->get_primitive_tc($CORBA::TCKind::tk_short);

# initialize the CORBA::Any pseudo-object with this value and typecode
$asset->insert($value, $tc);

# pass the CORBA::Any reference as a parameter
$col = $acct->collateral($asset);

# do something with the returned CORBA::Any reference
$ctc = $col->type();
if ($ctc->kind() == $CORBA::TCKind::tc_string) {
    # get string value from the CORBA::Any pseudo-object referred to by $col
    $val = $col->extract();
    ...
}
...
```

string

IDL-to-Perl コンパイラは、境界がある IDL の string と境界のない IDL の string の両方を Perl のスカラー型にマップします。Perl のスカラー文字列には常に境界がないため、メソッドの実行のために生成された Perl コードは、境界のある string メソッドのパラメータに渡された文字列の値が許容される長さを超える場合、その文字列の値を切り捨てます。

例

```
// IDL
module finance {
    interface account {
        double totaldeposits(in string frombankid, in string<8> date);
        ...
    };
    interface bank {
        account newaccount(in string name);
        account getaccount(in string name);
        ...
    };
    ...
};
```

Perl クライアントの使用方法

```
# Perl client
use finance;
...
$bank = finance::bank->bind();
$acct = $bank->getaccount("Joseph P. Shmuck");

# the date string passed below will be truncated to 8 characters,
# lopping off " extra chars"
$totaldeposits = $acct->totaldeposits("90-7005", "19980106 extra chars");
...
```

配列と sequence

IDL-to-Perl コンパイラは、IDL の配列と IDL の sequence を Perl の配列リファレンスにマップします。

例

```
// IDL
module finance {
    interface account {
        ...
```

```
    };
    interface bank {
        long getaccounts(out sequence<account> accounts);
        account newjointaccount(in sequence<string, 2> names);
        ...
    };
    ...
};
```

Perl クライアントの使用方法

```
# Perl client
use finance;
...
$bank = finance::bank->bind();

# pass reference to the array which getaccounts is to populate
$numaccts = $bank->getaccounts($accts);
for ($i = 0; $i < $numaccts; $i++) {
    $account = $accts->[$i];
    ...
}

# construct and pass an array reference to newjointaccount
$names = [ "Joseph P. Shmuck", "Josephine Q. Public" ];
$jointacct = $bank->newjointaccount($names);
...
```

演算

IDL-to-Perl コンパイラは、interface で宣言された IDL の演算をその interface を実現するパッケージ内の Perl メソッドにマップします。前述の「interface」の例を参照してください。

attribute

IDL-to-Perl コンパイラは、interface で宣言された IDL の attribute をオーバーロードされた Perl メソッドにマップします。Perl メソッドの名前は、その interface を実現するパッケージ内の attribute の名前をベースにして付けられます。そのメソッドをコールする方法は 2 つあります。

- 引数なしでコールされた場合、メソッドは attribute の現在の値を返します。
- 引数を 1 つ使用してコールされた場合、メソッドは attribute の値を引数の値に設定します。

例

```
// IDL
module finance{
    interface account {
```

```
        attribute double interestrate;
        ...
    };
    interface bank {
        account newaccount(in string name);
        account getaccount(in string name);
        ...
    }
    ...
};
```

Perl クライアントの使用方法

```
# Perl client
use finance;
...
$bank = finance::bank->bind();
$acct = $bank->getaccount("Joseph P. Shmuck");
$oldrate = $acct->interestrate();
$acct->interestrate($oldrate + 0.005);
```

列挙（enum）型

enum 型については、IDL-to-Perl コンパイラはスカラー変数を宣言するパッケージを生成します。スカラー変数の名前は、その型の enum 定数をベースにして付けられます。これらのスカラー変数には、0 から始まり増加する整数値が割り当てられます。パッケージ名はその enum 型の IDL 名と同じなので、その enum 型は Perl パッケージにマップされます。enum 型の変数は、Perl のスカラーにマップされます。

例

```
// IDL
module finance {
    interface account {
        enum trans_type { deposit, withdrawl };
        attribute trans_type lasttranstype;
        ...
    };
    interface bank {
        account newaccount(in string name);
        account getaccount(in string name);
        ...
    }
    ...
};
```

Perl クライアントの使用方法

```
# Perl client
use finance;
...
$bank = finance::bank->bind();
$acct = $bank->getacount("Joseph P. Shmuck");
$lasttranstype = $acct->lasttranstype();

# test value against an enumerator
if ($lasttranstype == $finance::account::trans_type::deposit) {
    ...
}
```

構造体

IDL の構造体については、IDL-to-Perl コンパイラがその構造体の名前をベースにした名前の Perl パッケージを自動的に生成します。この構造体の型の変数は、ハッシュのリファレンスにマップされます。ハッシュには、構造体のエレメントの名前をベースにした名前のキーが含まれます。

例

```
// IDL
module finance {
    interface account
        enum trans_type { deposit, withdrawl };
        struct transaction {
            string<8> date;
            trans_type type;
            double amount;
        };
        short do_transaction(inout transaction trans);
        ...
    };
    interface bank {
        account newaccount(in string name);
        account getaccount(in string name);
        ...
    }
    ...
};
```

Perl クライアントの使用方法

```
# Perl client
use finance;
```

```
...
$bank = finance::bank->bind();
$acct = $bank->getaccount("Jospeh P. Shmuck");

# construct a hash reperesenting a variable of IDL type transaction
$trans = {};
$trans->{date} = "19980106";
$trans->{type} = $finance::account::trans_type::withdrawl;
$trans->{amount} = 40.0;

# pass a reference to this hash as a parameter to an operation
$status = $acct->do_transaction($trans);
...
```

union

注意： 現在、union の Perl バインディングはインプリメントされていません。

typedef

IDL の typedef は、他の IDL 型の別名である新しい型を定義します。これは、typedef または IDL の基本型です。IDL の typedef は、この新しい型を実現する Perl パッケージにマップされます。このようなマッピングは、通常、ランタイム・システムによって引数のマーシャル / アンマーシャルに使用されます。typedef の変数は、別名を持つ展開された基本型のルールに従ってマップされます。

例

```
// IDL
module finance {
    interface account {
        typedef enum trans_type { deposit, withdrawl } trans_t;
        attribute trans_t lasttranstype;
        ...
    };
    interface bank {
        account newaccount(in string name);
        account getaccount(in string name);
        ...
    }
    ...
};
```

Perl クライアントの使用方法

```
# Perl Client
use finance;
...
$bank = finance::bank->bind();
$lasttranstype = $acct->lasttranstype();

# test value against an enumerator--
# must specify the base type here rather than the defined type
if ($lasttranstype == $finance::account::trans_type::deposit) {
    ...
}
...
```

exception

> **注意：** ここで説明するインプリメンテーションは、将来のリリースでは本来の例外メカニズムに置き換えられます。

IDL の exception は、IDL の構造体と非常によく似た形でマップされます。IDL の exception については、IDL-to-Perl コンパイラがその exception の名前をベースにした名前の Perl パッケージを生成します。この exception 型の変数は、ハッシュのリファレンスにマップされます。ハッシュには、exception のエレメントの名前をベースにした名前のキーが含まれます。

IDL の処理では、通常の戻り値のかわりに例外のリファレンスを返すことによって例外が発生します。Perl からこのような処理をコールする場合の正しい方法は、`Oracle::hlpr::Excp` クラスのクラス・メソッドを使用して、戻り値が例外であるかどうかをテストすることです。例外である場合、`Oracle::hlpr::Excp` クラスの別のクラス・メソッドを使用して、例外を再発行します。

現在、例外の型を特定したり、例外を出力する方法はありません。

このリリースでは、拡張された例外処理のサポートはありません。例外パッケージは、将来のリリースで提供される予定です。

CORBA サーバー・オブジェクトから発生するユーザー定義例外のアンマーシャルは、このリリースにはありません。ユーザーは、ユーザー定義の例外のフィールドにアクセスできません。この機能のサポートは、将来のリリースで追加される予定です。

CORBA のシステム例外は Perl にマップされる必要があります。これは将来のリリースで実現される予定です。

Oracle::hlpr::Excp には、次のクラス・メソッドがあります。

- isexcp()—1 つの引数付きでコールされ、その引数が例外オブジェクトのリファレンスである場合、このメソッドは True スカラー値を返します。
- throw()—1 つの引数付きでコールされ、その引数が参照している例外を発行します。引数は、例外オブジェクトのリファレンスである必要があります。
- rethrow()— 引数なしでコールされ、最後の例外を再発行します。

例

```
// IDL
module finance {
    interface account {
        typedef enum trans_type { deposit, withdrawl } trans_t;
        attribute trans_t lasttranstype;
        ...
    };
    interface bank {
        exception badaccount {};
        account newaccount(in string name);
        account getaccount(in string name) raises(badaccount);
        ...
    }
    ...
};
```

Perl クライアントの使用方法

```
# Perl Client
use finance;
use Oracle::hlpr::Excp;
...
$bank = finance::bank->bind();

$acct = $bank->getaccount("Joseph P. Shmuck");
if (Oracle::hlpr::Excp->isexcp($acct)) {
    Oracle::hlpr::Excp->rethrow();
}
```

8

Perl クライアントの CORBA 擬似オブジェクト API

Oracle Application Server には、Perl クライアントの Static Invocation Interface（SII）をサポートするために必要な、CORBA の擬似オブジェクト・インタフェース用の Perl バインディングが用意されています。この章では Perl のサンプル・コードを使用して、サポートされている擬似オブジェクト・インタフェースのコールについて説明します。この章では、ほとんどのインタフェースについて、そのコール方法については説明しません。これについては、Object Management Group の Web サイト **http://www.omg.org** の「The Common Object Request Broker: Architecture and Specification」で定義されています。ただし、一部のインタフェースは擬似オブジェクトの Perl インプリメンテーションに固有のものです。このような場合にはコール方法について説明します。

> **注意：** Oracle Application Server には、これらの擬似オブジェクト用の完全な Perl バインディングは用意されていません。後の項で、バインディングが用意されている擬似オブジェクト・インタフェースのサブセットと、Perl バインディングが独自に定義する特殊なインタフェースについて説明します。

内容

この章では、次の CORBA 擬似オブジェクト・インタフェースの Perl バインディングについて説明します。

Object

次の Object インタフェースのサブセットについては、Perl バインディングが用意されています。

```
// PIDL
module CORBA {
    interface Object {
        // instance methods
        Object duplicate();
        void release();
        boolean is_a(in string logical_type_id);
        boolean non_existent();
        boolean is_equivalent(in Object other_object);
    };
};
```

これらの操作の Perl バインディングは、先頭にアンダースコア（_）が付いた同じ名前を使用します。たとえば、duplicate() は Perl の _duplicate() にマップされます。

インスタンス・メソッド

次の例では、*$obj* はすべて CORBA オブジェクトのリファレンスです。

duplicate()

```
// PIDL
Object duplicate();
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::Object;
...
$dup = $obj->_duplicate();
```

release()

```
// PIDL
void release();
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::Object;
...
$obj->_release();
```

is_a()

```
// PIDL
boolean is_a(in string logical_type_id);
```

Perl からのサンプル・コール

この例では、*$logical_id* は文字列の値を持つ Perl スカラー型です。

```
# Perl client
use CORBA::Object;
...
if ($obj->_is_a($logical_id)) {
    ...
}
```

non_existent()

```
// PIDL
boolean non_existent();
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::Object;
...
if ($obj->_non_existent()) {
    ...
}
```

is_equivalent()

```
// PIDL
boolean is_equivalent(in Object other_object);
```

Perl からのサンプル・コール

この例では、*$other_object* は CORBA オブジェクトのリファレンスです。

```
# Perl client
use CORBA::Object;
...
if ($obj->_is_equivalent($other_object)) {
    ...
}
```

ORB

次の ORB インタフェースのサブセットについて、Perl バインディングが用意されています。

```
// PIDL
module CORBA {
    interface ORB {
        typedef string ObjectId;
        typedef sequence<ObjectId> ObjectIdList;

        exception InvalidName{};

        // class methods
        ObjectIdList list_initial_services();
        Object resolve_initial_references(in ObjectID identifier)
            raises(InvalidName);

        string object_to_string(in Object obj);
        Object string_to_object(in string str);

        // interfaces defined by the Perl bindings--
        // use the init() operations in place of CORBA::ORB_init()
        ORB init();
        ORB init(inout sequence<string> argv);

        // instance methods
        Object bind(string object_id);
        void term();
    };
};
```

クラス・メソッド

list_initial_services()

```
// PIDL
ObjectIdList list_initial_services();
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::ORB;
...
$servlist = CORBA::ORB->list_initial_services();
$numservs = scalar(@$servlist);
for ($i = 0; $i < $numservs; $i++) {
    $service = $serlist->[$i];
    ...
}
```

resolve_initial_references()

```
// PIDL
Object resolve_initial_references(in ObjectID identifier)
    raises(InvalidName);
```

Perl からのサンプル・コール

この例では、*$obj_id* は IDL 型の ObjectID の値の文字列を持つスカラー型です。この値は、list_initial_services() から返される値の 1 つです。

```
# Perl client
Fuse Oracle::hlpr::Excp;
...
$obj = CORBA::ORB->resolve_initial_references($obj_id);
if (Oracle::hlpr::Excp->isexcp($obj)) {
    Oracle::hlpr::Excp->throw($obj);
}
```

object_to_string()

```
// PIDL
string object_to_string(in Object obj);
```

Perl からのサンプル・コール

この例では、*$obj* は CORBA オブジェクトのリファレンスです。

```
# Perl client
use CORBA::ORB;
...
$str = CORBA::ORB->object_to_string($obj);
```

string_to_object()

```
// PIDL
Object string_to_object(in string str);
```

Perl からのサンプル・コール

この例では、*$string* は文字列の値を持つスカラー型です。

```
# Perl client
use CORBA::ORB;
...
$obj = CORBA::ORB->string_to_object($string);
```

init()

> **注意：** LiveHTML ドキュメント内からこのメソッドをコールしないでください。スクリプトが LiveHTML ドキュメント内から実行されると、そのスクリプトは自動的に実行中の ORB にアクセスします。

この演算を使用して、Object Request Broker（ORB）の初期化と ORB 上のサービスの実行を行います。

```
// PIDL
ORB init();
ORB init(inout sequence<string> argv);
```

2 番目の形式の `init()` を使用する場合、`argv` 配列の先頭のエレメントは有効な Java の CLASSPATH 値である必要があります。この値には、CORBA の擬似オブジェクトの Java バインディングの場所を 1 つまたは複数指定する必要があります。必ず、CLASSPATH にディレクトリ `$ORAWEB_HOME/../cartx/livehtml/stubs/java/` が含まれている必要があります。IDL-to-Perl コンパイラは Java バインディングを生成し、これを Perl バインディングが ORB オブジェクトおよび CORBA オブジェクト上の演算を実行するために使用するため、これは必須です。

Perl からのサンプル・コール

```
# Perl client
use CORBA::ORB;
...
$orb = CORBA::ORB->init();
```

インスタンス・メソッド

bind()

CORBA オブジェクトにバインドされるメソッドは、引数によって識別されます。

```
// PIDL
Object bind(string object_id);
```

Perl からのサンプル・コール

この例では、

- *`$orb`* は CORBA ORB 擬似オブジェクトのリファレンスです。
- *`$object_id`* は文字列の値を持つスカラー型です。

```
# Perl client
use CORBA::ORB;
...
$obj = $orb->bind($object_id)
```

term()

> **注意：** LiveHTML ドキュメント内からこのメソッドをコールしないでください。LiveHTML ドキュメント内からスクリプトを実行する際に、LiveHTML カートリッジが使用している ORB を終了しないでください。

このメソッドを使用して ORB を終了します。このメソッドからリターンした後は、CORBA オブジェクトは実行できません。

```
// PIDL
void term();
```

Perl からのサンプル・コール

この例では、*$orb* は CORBA の ORB 擬似オブジェクトのリファレンスです。

```
# Perl client
use CORBA::ORB;
...
$orb->term();
```

Any

次の Any インタフェースのサブセットについて、Perl バインディングが用意されています。

```
// PIDL
module CORBA {
    interface Any {
        // instance methods
        TypeCode type();
        void insert(in any value, in TypeCode type);
        any extract();    // actually returns a value of type
                          // set by insert()
    };
};
```

インスタンス・メソッド

次の例では、*$any* は CORBA の Any 擬似オブジェクトのリファレンスです。

type()

```
// PIDL
TypeCode type();
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::Any;
...
$type = $any->type();
```

insert()

この PIDL では、*anyval* は type パラメータでエンコードされた型を表します。

```
// PIDL
void insert(in anyval value, in TypeCode type);
```

Perl からのサンプル・コール

この例では、

- *$value* は、*$type* で示された型の値、またはその値に対するリファレンスです。
- *$type* は、CORBA の TypeCode 擬似オブジェクトのリファレンスです。

> **注意：** この値によって、CORBA::TypeCode の create_*_tc() メソッドを使用して作成された typecode を参照しないでください。CORBA オブジェクトの _type() メソッドによって返される typecode、または CORBA::TypeCode の get_primitive_tc() メソッドによって返されるプリミティブ型の typecode を使用できます。

```
# Perl client
use CORBA::Any;
use CORBA::TypeCode;
...

$any->insert($value, $type);
```

extract()

この PIDL では、*anyval* は、擬似オブジェクトを初期化した insert() コールによって指定された型を表します。

```
// PIDL
anyval extract();    // actually returns a value of type set by insert()
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::Any;
use CORBA::TypeCode;
use CORBA::TCKind;
...
$type = $any->type();
$val = $any->extract();
$kind = $type->kind();
if ($kind == $CORBA::TCKind::tk_short) {
    ...
}
elsif ($kind == $CORBA::TCKind::tk_long) {
    ...
}
```

一般的にこの例と同じ様な状況では、どの $kind になる可能性があるかがコンテキストによって示されます。通常、可能性がある TCKind 値をすべてテストする必要はありません。

TypeCode

CORBA の仕様では、ここにリストされているクラス・メソッドは interface ORB で宣言されます。ただし Perl バインディングは、次に示すように、これらのメソッドを interface TypeCode のクラス・メソッドとしてインプリメントしています。

```
// PIDL
module CORBA {
    struct StructMember {
        string name;
        TypeCode type;
        IDLType type_def; // currently not used
    }
    typedef sequence<StructMember> StructMemberSeq;

    typedef sequence<string> EnumMemberSeq;

    interface TypeCode {
        // class methods
        Any create_any();
        TypeCode create_struct_tc (
            in string repository_id,
            in string type_name,
            in StructMemberSeq members
        );
        TypeCode create_enum_tc (
            in string repository_id,
```

```
    in string type_name,
    EnumMemberSeq members
);
TypeCode create_alias_tc (
    in string repository_id,
    in string type_name,
    in TypeCode original_type
);
TypeCode create_exception_tc (
    in string repository_id,
    in string type_name,
    in StructMemberSeq members
);
TypeCode create_interface_tc (
    in string repository_id,
    in string type_name
);
TypeCode create_string_tc (
    in unsigned long bound
);
TypeCode create_wstring_tc (
    in unsigned long bound
);
TypeCode create_sequence_tc (
    in unsigned long bound,
    in TypeCode type
);
TypeCode create_array_tc (
    in unsigned long length,
    in TypeCode type
);

// class method defined by the Perl bindings
TypeCode get_primitive_tc(TCKind kind);

// exceptions raised by instance methods
exception Bounds {};
exception BadKind {};

// instance methods
boolean equal(in TypeCode tc);
TCKind kind();
string id() raises(BadKind);
string name() raises(BadKind);
unsigned long member_count() raises(BadKind);
string member_name(
    in unsigned long index
```

```
        ) raises(BadKind, Bounds);
        TypeCode member_type(
            in unsigned long index
        ) raises(BadKind, Bounds);
        any member_label(
            in unsigned long index
        ) raises(BadKind, Bounds);
        TypeCode discriminator_type() raises(BadKind);
        long default_index() raises(BadKind);
        unsigned long length() raises(BadKind);
        TypeCode content_type() raises(BadKind);

        // instance method defined by the Perl bindings
        TypeCode orig_type();
    };
};
```

クラス・メソッド

CORBA の仕様では、これらのメソッドを `interface ORB` で宣言します。ただし Perl バインディングは、これらのメソッドを `interface TypeCode` のクラス・メソッドとしてインプリメントしています。

次の例では、*`$repository_id`* と *`$type_name`* は文字列の値を持つスカラー型です。

create_any()

```
// PIDL
Any create_any();
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
...
$any = CORBA::TypeCode->create_any();
```

create_struct_tc()

```
// PIDL
TypeCode create_struct_tc (
    in string repository_id,
    in string type_name,
    in StructMemberSeq members
);
```

Perl からのサンプル・コール

この例では、

- *$member_name1* と *$member_name2* は、文字列の値を持つスカラー型です。
- *$member_type1* と *$member_type2* は、CORBA の TypeCode 擬似オブジェクトのリファレンスです。

```
# Perl client
use CORBA::TypeCode;
...
# construct references to hashes defining structure members
$member1 =
  { "name" => $member_name1, "type" => $member_type1, "type_def" => "" };
$member2 =
  { "name" => $member_name2, "type" => $member_type2, "type_def" => "" };
# repeat for additional structure members

# construct reference to array of members, which defines the structure
$members = [ $member1, $member2 ];
$newtypecode =
  CORBA::TypeCode->create_struct_tc($repository_id, $type_name, $members);
```

create_enum_tc()

```
// PIDL
TypeCode create_enum_tc (
    in string repository_id,
    in string type_name,
    EnumMemberSeq members
);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
...
# construct a reference to an array containing the new enum value names
$enumvals = [ "val1", "val2", "val3" ];
$newtypecode =
  CORBA::TypeCode->create_enum_tc($repository_id, $type_name, $enumvals);
```

create_alias_tc()

```
// PIDL
TypeCode create_alias_tc (
    in string repository_id,
    in string type_name,
    in TypeCode original_type
);
```

Perl からのサンプル・コール

この例では、*$original_type* は CORBA の TypeCode 擬似オブジェクトのリファレンスです。

```
# Perl client
use CORBA::TypeCode;
...
$newtypecode =
  CORBA::TypeCode->create_alias_tc($repository_id, $type_name,
  $original_type);
```

create_exception_tc()

```
// PIDL
TypeCode create_exception_tc (
    in string repository_id,
    in string type_name,
    in StructMemberSeq members
);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
...
# construct references to hashes defining exception members
$member1 =
  { "name" => $member_name1, "type" => $member_type1, "type_def" => "" };
$member2 =
  { "name" => $member_name2, "type" => $member_type2, "type_def" => "" };
# repeat for additional exception members

# construct reference to array of members, which defines the exception
$members = [ $member1, $member2 ];
$newtypecode =
  CORBA::TypeCode->create_struct_tc($repository_id, $type_name, $members);
```

create_interface_tc()

```
// PIDL
TypeCode create_interface_tc (
    in string repository_id,
    in string type_name
);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
...
$newtypecode = CORBA::TypeCode->($repository_id, $type_name);
```

create_string_tc()

```
// PIDL
TypeCode create_string_tc (
    in unsigned long bound
);
```

Perl からのサンプル・コール

この例では、*$bound* は符号なし長整数型の値を持つスカラー型です。

```
# Perl client
use CORBA::TypeCode;
...
$newtypecode = CORBA::TypeCode->create_string_tc($bound);
```

create_wstring_tc()

```
// PIDL
TypeCode create_wstring_tc (
    in unsigned long bound (
);
```

Perl からのサンプル・コール

この例では、*$bound* は符号なし長整数型の値を持つスカラー型です。

```
# Perl client
use CORBA::TypeCode;
...
$newtypecode = CORBA::TypeCode->create_wstring_tc($bound);
```

create_sequence_tc()

```
// PIDL
TypeCode create_sequence_tc (
    in unsigned long bound,
    in TypeCode type
);
```

Perl からのサンプル・コール

この例では、

- *$bound* は符号なし長整数型の値を持つスカラー型です。
- *$type* は、CORBA の TypeCode 擬似オブジェクトのリファレンスです。

```
# Perl client
use CORBA::TypeCode;
...
$newtypecode = CORBA::create_sequence_tc($bound, $type);
```

create_array_tc()

```
// PIDL
TypeCode create_array_tc (
    in unsigned long length,
    in TypeCode type
);
```

Perl からのサンプル・コール

この例では、

- *$length* は符号なし長整数型の値を持つスカラー型です。
- *$type* は、CORBA の TypeCode 擬似オブジェクトのリファレンスです。

```
# Perl client
use CORBA::TypeCode;
...
$newtypecode = CORBA::create_array_tc($length, $type);
```

get_primitive_tc()

```
// PIDL
TypeCode get_primitive_tc(TCKind kind);
```

Perl からのサンプル・コール

この例では、*$kind* は enum TCKind で定義された値を持つスカラー型です。
後述の「TCKind」を参照してください。

```
# Perl client
use CORBA::TypeCode;
use CORBA::TCKind;
...
$typecode = CORBA::get_primitive_tc($kind);
```

インスタンス・メソッド

次の例では、*$typecode* は CORBA の TypeCode 擬似オブジェクトのリファレンスです。

equal()

```
// PIDL
boolean equal(in TypeCode tc);
```

Perl からのサンプル・コール

この例では、*$other_typecode* は CORBA の TypeCode 擬似オブジェクトのリファレンスです。

```
# Perl client
use CORBA::TypeCode;
...
if ($typecode->equal($other_typecode)) {
    ...
}
```

kind()

```
// PIDL
TCKind kind();
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
...
$kind = $typecode->kind();
if ($kind == $CORBA::TCKind::tk_short) {
    ...
}
```

id()

```
// PIDL
string id() raises(BadKind);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$id = $typecode->id();
if (Oracle::hlpr::Excp->isexcp($id)) {
    Oracle::hlpr::Excp->throw($id);
}
```

name()

```
// PIDL
string name() raises(BadKind);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$name = $typecode->name();
if (Oracle::hlpr::Excp->isexcp($name)) {
    Oracle::hlpr::Excp->throw($name);
}
```

member_count()

```
// PIDL
unsigned long member_count() raises(BadKind);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$num_members = $typecode->member_count();
if (Oracle::hlpr::Excp->isexcp($num_members)) {
    Oracle::hlpr::Excp->throw($num_members);
}
```

member_name()

```
// PIDL
string member_name(
    in unsigned long index
) raises(BadKind, Bounds);
```

Perl からのサンプル・コール

この例では、*$index* は符号なし長整数型の値を持つスカラー値です。

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$mem_name = $typecode->member_name($index);
if (Oracle::hlpr::Excp->isexcp($mem_name)) {
    Oracle::hlpr::Excp->throw($mem_name);
}
```

member_type()

```
// PIDL
TypeCode member_type(
    in unsigned long index
) raises(BadKind, Bounds);
```

Perl からのサンプル・コール

この例では、*$index* は符号なし長整数型の値を持つスカラー値です。

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$mem_type = $typecode->member_type($index);
if (Oracle::hlpr::Excp->isexcp($mem_type)) {
    Oracle::hlpr::Excp->throw($mem_type);
}
```

member_label()

```
// PIDL
any member_label(
    in unsigned long index
) raises(BadKind, Bounds);
```

Perl からのサンプル・コール

この例では、*$index* は符号なし長整数型の値を持つスカラー値です。

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$label = $typecode->member_label($index);
if (Oracle::hlpr::Excp->isexcp($label)) {
    Oracle::hlpr::Excp->throw($label);
}
```

discriminator_type()

```
// PIDL
TypeCode discriminator_type() raises(BadKind);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$tc = $typecode->discriminator_type();
if (Oracle::hlpr::Excp->isexcp($tc)) {
    Oracle::hlpr::Excp->throw($tc);
}
```

default_index()

```
// PIDL
long default_index() raises(BadKind);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$def_index = $typecode->default_index();
if (Oracle::hlpr::Excp->isexcp($def_index)) {
    Oracle::hlpr::Excp->throw($def_index);
}
```

length()

```
// PIDL
unsigned long length() raises(BadKind);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$len = $typecode->length();
if (Oracle::hlpr::Excp->isexcp($len)) {
    Oracle::hlpr::Excp->throw($len);
}
```

content_type()

```
// PIDL
TypeCode content_type() raises(BadKind);
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
use Oracle::hlpr::Excp;
...
$tc = $typecode->content_type();
if (Oracle::hlpr::Excp->isexcp($tc)) {
    Oracle::hlpr::Excp->throw($tc);
}
```

orig_type()

```
// PIDL
TypeCode orig_type();
```

Perl からのサンプル・コール

```
# Perl client
use CORBA::TypeCode;
...
$tc = $typecode->orig_type();
```

TCKind

enum TCKind には、Perl バインディングが用意されています。

```
// PIDL
module CORBA {
    enum TCKind {
        tk_null, tk_void,
        tk_short, tk_long, tk_ushort, tk_ulong,
        tk_float, tk_double, tk_boolean, tk_char,
        tk_octet, tk_any, tk_TypeCode, tk_Principal, tk_objref,
        tk_struct, tk_union, tk_enum, tk_string,
        tk_sequence, tk_array, tk_alias, tk_except,
        tk_longlong, tk_ulonglong, tk_longdouble,
        tl_wchar, tk_wstring, tk_fixed, tk_byte
    };
};
```

Perl バインディングは tk_byte の値を定義します。これは、CORBA の仕様では宣言されません。

TCKind の使用例は、前述の TypeCode 擬似オブジェクトの kind() メソッドを参照してください。

9

IDL-to-Perl コンパイラのサンプル出力

この章では、IDL-to-Perl コンパイラが次のサンプル IDL のコンパイル中に生成するディレクトリ構造と Perl のモジュール・ファイルについて説明します。

内容

サンプル IDL

この IDL は、第 7 章「Perl スクリプトからの CORBA オブジェクトへのアクセス」で使用した例のサマリーです。

```
// IDL

module finance {
    const long L = 3;

    interface account {
        const double minbalance = 500.0;
        typedef enum trans_type { deposit, withdrawl } trans_t;

        struct transaction {
            string<8> date;
            trans_t type;
            double amount;
        };

        attribute double interestrate;
        attribute trans_t lasttranstype;
```

```
        double getbalance();
        double totaldeposits(in string frombankid, in string<8> date);
        short do_transaction(inout transaction trans);

        any collateral(in any asset);
    };

    interface checkingaccount : account {

    };

    interface bank {
        const short stuff = 3;
        typedef sequence<string, 2> stringseq;
        typedef sequence<account> accountseq;

        account newaccount(in string name);
        account newjointaccount(in stringseq names);
        checkingaccount newchecking(in string name);

        exception badaccount {};

        account getaccount(in string name) raises(badaccount);
        long getaccounts(out accountseq accounts) raises(badaccount);
        checkingaccount getchecking(in string name) raises(badaccount);
    };
};

module outer {
    module inner {
        const short thing = 3;
    };
};
```

生成されたファイルのディレクトリ構造

次のコマンド・ラインは、ファイル ex.idl に含まれている前述の IDL のコンパイルに使用されます。

```
perlidlc ex.idl
```

このコマンドにより、ディレクトリ $ORAWEB_HOME/../cartx/livehtml/stubs/perl/ に次のファイルとディレクトリが作成されます。ディレクトリは、UNIX スタイルのスラッシュ '/' を後に付けて示しています。

```
finance/
    account/
        trans_t.pm
        trans_type.pm
        transaction.pm
    account.pm
    bank/
        accountseq.pm
        badaccount.pm
        stringseq.pm
    bank.pm
    checkingaccount.pm
finance.pm
outer/
    inner.pm
outer.pm
```

生成されたファイルのリスト

次のファイルは、前の項に示された順番にリストされます。

finance/account/trans_t.pm

```
package finance::account::trans_t;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;

sub _type {
if(!defined($finance::account::trans_t::alias_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass (
  "finance/accountPackage/trans_tHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::account::trans_t";

    my $origtc = finance::account::trans_type->_type();
    $finance::account::trans_t::alias_tc = CORBA::TypeCode->create_alias_tc($id,
  $name, $origtc);
    $finance::account::trans_t::alias_tc->jClass(
  "finance/accountPackage/trans_t");
    $finance::account::trans_t::alias_tc->perlClass(
  "finance::account::trans_t");
    $finance::account::trans_t::alias_tc->JNItype(
```

```
  "Lfinance/accountPackage/trans_type;");
}

return $finance::account::trans_t::alias_tc;
}

1;
```

finance/account/trans_type.pm

```
package finance::account::trans_type;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;

*deposit = ¥0;
*withdrawl = ¥1;

sub _type {
if(!defined($finance::account::trans_type::enum_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass (
      "finance/accountPackage/trans_typeHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::account::trans_type";

    my $members = [];
    $members->[0] = {};
    $members->[0]->{name} = "deposit";
    $members->[1] = {};
    $members->[1]->{name} = "withdrawl";
    $finance::account::trans_type::enum_tc = CORBA::TypeCode->create_enum_tc(
      $id, $name, $members);
    $finance::account::trans_type::enum_tc->jClass(
      "finance/accountPackage/trans_type");
    $finance::account::trans_type::enum_tc->perlClass(
      "finance::account::trans_type");
    $finance::account::trans_type::enum_tc->JNItype(
      "Lfinance/accountPackage/trans_type;");
}


return $finance::account::trans_type::enum_tc;
}

1;
```

finance/account/transaction.pm

```
package finance::account::transaction;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;

use string;
use finance::account::trans_t;

sub _type {
if(!defined($finance::account::transaction::struct_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass (
      "finance/accountPackage/transactionHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::account::transaction";

    my $members = [];
    $members->[0] = {};
    $members->[0]->{name} = "date";
    $members->[0]->{tc} = string->_type();
    $members->[1] = {};
    $members->[1]->{name} = "type";
    $members->[1]->{tc} = finance::account::trans_t->_type();
    $members->[2] = {};
    $members->[2]->{name} = "amount";
    $members->[2]->{tc} = Oracle::hlpr::prim_double->_type();
    $finance::account::transaction::struct_tc =
      CORBA::TypeCode->create_struct_tc($id, $name, $members);
    $finance::account::transaction::struct_tc->jClass(
      "finance/accountPackage/transaction");
    $finance::account::transaction::struct_tc->perlClass(
      "finance::account::transaction");
    $finance::account::transaction::struct_tc->JNItype(
      "Lfinance/accountPackage/transaction;");
}

return $finance::account::transaction::struct_tc;
}

1;
```

finance/account.pm

```
package finance::account;

use finance::account::trans_type;
use finance::account::trans_t;
use finance::account::transaction;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use CORBA::Object;
use Oracle::hlpr::Marshall;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;


@ISA = qw ( CORBA::Object );

sub _type {
if(!defined($finance::account::intf_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass ("finance/accountHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::account";

    $finance::account::intf_tc = CORBA::TypeCode->create_interface_tc($id,
      $name);
    $finance::account::intf_tc->jClass("finance/account");
    $finance::account::intf_tc->perlClass("finance::account");
    $finance::account::intf_tc->JNItype("Lfinance/account;");
}

return $finance::account::intf_tc;
}

*minbalance = ¥500.000000;
sub interestrate {
my $self = shift; my $jClass;
if (scalar(@_)) {
use Oracle::hlpr::Primitives;
my $interestrateJava = Oracle::hlpr::Marshall->marshal( $_[0],
  Oracle::hlpr::prim_double->_type() );
my $ret = $self->{javaObj}->interestrate("$double", $interestrateJava, "$void");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
return ;
} else {
my $ret = $self->{javaObj}->interestrate("$double");
return Oracle::hlpr::Excp->throw($ret)
```

```
        if (Oracle::hlpr::Excp->isexcp($ret));
use Oracle::hlpr::Primitives;
$retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  Oracle::hlpr::prim_double->_type() );
return $retNew;
}
}

sub lasttranstype {
my $self = shift; my $jClass;
if (scalar(@_)) {
use finance::account::trans_t;
my $lasttranstypeJava = Oracle::hlpr::Marshall->marshal( $_[0],
  finance::account::trans_t->_type() );
my $ret = $self->{javaObj}->lasttranstype(
  "Lfinance/accountPackage/trans_type;", $lasttranstypeJava, "$void");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
return ;
} else {
my $ret = $self->{javaObj}->lasttranstype(
  "Lfinance/accountPackage/trans_type;");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use finance::account::trans_t;
$retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  finance::account::trans_t->_type() );
return $retNew;
}
}

sub getbalance {
my $self = shift; my $jClass;
my $ret = $self->{javaObj}->getbalance("$double");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use Oracle::hlpr::Primitives;
$retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  Oracle::hlpr::prim_double->_type() );
return $retNew;
}

sub totaldeposits {
my $self = shift; my $jClass;
use Oracle::hlpr::Primitives;
my $frombankidJava = Oracle::hlpr::Marshall->marshal( $_[0],
  Oracle::hlpr::prim_string->_type() );
use string;
my $dateJava = Oracle::hlpr::Marshall->marshal( $_[1], string->_type() );
```

```
my $ret = $self->{javaObj}->totaldeposits("$String", $frombankidJava, "$String",
  $dateJava, "$double");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use Oracle::hlpr::Primitives;
$retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  Oracle::hlpr::prim_double->_type() );
return $retNew;
}

sub do_transaction {
my $self = shift; my $jClass;
use finance::account::transaction;
my $transJava = Oracle::hlpr::Marshall->marshal( $_[0],
  finance::account::transaction->_type() );
$jClass = Oracle::Java::ClassLdr->loadClass (
  "finance/accountPackage/transactionHolder");
$transHldr = Oracle::Java::Object->new ($jClass);
$transHldr->_set_field ("value",
  "Lfinance/accountPackage/transaction;", $transJava);
my $ret = $self->{javaObj}->do_transaction(
  "Lfinance/accountPackage/transactionHolder;", $transHldr, "$short");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
$transOut = $transHldr->_get_field ("value",
  "Lfinance/accountPackage/transaction;");
use finance::account::transaction;
$_[0] = Oracle::hlpr::Marshall->unmarshal( $transOut,
  finance::account::transaction->_type() );
use Oracle::hlpr::Primitives;
$retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  Oracle::hlpr::prim_short->_type() );
return $retNew;
}

sub collateral {
my $self = shift; my $jClass;
use CORBA::Any;
my $assetJava = Oracle::hlpr::Marshall->marshal( $_[0], CORBA::Any->_type() );
my $ret = $self->{javaObj}->collateral("$CORBA::Any::JNItype", $assetJava,
  "$CORBA::Any::JNItype");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use CORBA::Any;
$retNew = Oracle::hlpr::Marshall->unmarshal( $ret, CORBA::Any->_type() );
return $retNew;
}

1;
```

finance/bank/accountseq.pm

```
package finance::bank::accountseq;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;

sub _type {
if(!defined($finance::bank::accountseq::alias_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass (
  "finance/bankPackage/accountseqHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::bank::accountseq";

    my $array_tc = CORBA::TypeCode->create_sequence_tc(0,
  finance::account->_type());
    $array_tc->JNItype("[Lfinance/account;");
    my $origtc = $array_tc;
    $finance::bank::accountseq::alias_tc = CORBA::TypeCode->create_alias_tc($id,
  $name, $origtc);
    $finance::bank::accountseq::alias_tc->jClass(
  "finance/bankPackage/accountseq");
    $finance::bank::accountseq::alias_tc->perlClass("finance::bank::accountseq");
    $finance::bank::accountseq::alias_tc->JNItype("[Lfinance/account;");
}

return $finance::bank::accountseq::alias_tc;
}

1;
```

finance/bank/badaccount.pm

```
package finance::bank::badaccount;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;
```

```
sub _type {
if(!defined($finance::bank::badaccount::struct_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass (
      "finance/bankPackage/badaccountHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::bank::badaccount";

    my $members = [];
    $finance::bank::badaccount::struct_tc =
      CORBA::TypeCode->create_exception_tc($id, $name, $members);
    $finance::bank::badaccount::struct_tc->jClass(
      "finance/bankPackage/badaccount");
    $finance::bank::badaccount::struct_tc->perlClass(
      "finance::bank::badaccount");
    $finance::bank::badaccount::struct_tc->JNItype(
      "Lfinance/bankPackage/badaccount;");
}

return $finance::bank::badaccount::struct_tc;
}

1;
```

finance/bank/stringseq.pm

```
package finance::bank::stringseq;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;

sub _type {
if(!defined($finance::bank::stringseq::alias_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass (
  "finance/bankPackage/stringseqHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::bank::stringseq";

    my $array_tc = CORBA::TypeCode->create_sequence_tc(2,
  Oracle::hlpr::prim_string->_type());
    $array_tc->JNItype("[$String");
    my $origtc = $array_tc;
    $finance::bank::stringseq::alias_tc = CORBA::TypeCode->create_alias_tc(
  $id, $name, $origtc);
```

```
    $finance::bank::stringseq::alias_tc->jClass(
  "finance/bankPackage/stringseq");
    $finance::bank::stringseq::alias_tc->perlClass("finance::bank::stringseq");
    $finance::bank::stringseq::alias_tc->JNItype("[$String");
}

return $finance::bank::stringseq::alias_tc;
}

1;
```

finance/bank.pm

```
package finance::bank;

use finance::bank::stringseq;
use finance::bank::accountseq;
use finance::bank::badaccount;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use CORBA::Object;
use Oracle::hlpr::Marshall;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;

@ISA = qw ( CORBA::Object );

sub _type {
if(!defined($finance::bank::intf_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass ("finance/bankHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::bank";

    $finance::bank::intf_tc = CORBA::TypeCode->create_interface_tc($id, $name);
    $finance::bank::intf_tc->jClass("finance/bank");
    $finance::bank::intf_tc->perlClass("finance::bank");
    $finance::bank::intf_tc->JNItype("Lfinance/bank;");
}

return $finance::bank::intf_tc;
}

*stuff = ¥3;
sub newaccount {
```

```
my $self = shift; my $jClass;
use Oracle::hlpr::Primitives;
my $nameJava = Oracle::hlpr::Marshall->marshal( $_[0],
  Oracle::hlpr::prim_string->_type() );
my $ret = $self->{javaObj}->newaccount("$String", $nameJava,
  "Lfinance/account;");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use finance::account;
my $retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  finance::account->_type() );
return $retNew;
}

sub newjointaccount {
my $self = shift; my $jClass;
use finance::bank::stringseq;
my $namesJava = Oracle::hlpr::Marshall->marshal( $_[0],
  finance::bank::stringseq->_type() );
my $ret = $self->{javaObj}->newjointaccount("[$String", $namesJava,
  "Lfinance/account;");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use finance::account;
my $retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  finance::account->_type() );
return $retNew;
}

sub newchecking {
my $self = shift; my $jClass;
use Oracle::hlpr::Primitives;
my $nameJava = Oracle::hlpr::Marshall->marshal( $_[0],
  Oracle::hlpr::prim_string->_type() );
my $ret = $self->{javaObj}->newchecking("$String", $nameJava,
  "Lfinance/checkingaccount;");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use finance::checkingaccount;
my $retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  finance::checkingaccount->_type() );
return $retNew;
}

sub getaccount {
my $self = shift; my $jClass;
use Oracle::hlpr::Primitives;
```

```
my $nameJava = Oracle::hlpr::Marshall->marshal( $_[0],
  Oracle::hlpr::prim_string->_type() );
my $ret = $self->{javaObj}->getaccount("$String", $nameJava,
  "Lfinance/account;");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use finance::account;
my $retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  finance::account->_type() );
return $retNew;
}

sub getaccounts {
my $self = shift; my $jClass;
$jClass = Oracle::Java::ClassLdr->loadClass (
  "finance/bankPackage/accountseqHolder");
my $accountsHldr = Oracle::Java::Object->new ($jClass);
my $ret = $self->{javaObj}->getaccounts(
  "Lfinance/bankPackage/accountseqHolder;", $accountsHldr, "$int");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
my $accountsOut = $accountsHldr->_get_field ("value", "[Lfinance/account;");
use finance::bank::accountseq;
$_[0] = Oracle::hlpr::Marshall->unmarshal( $accountsOut,
  finance::bank::accountseq->_type() );
use Oracle::hlpr::Primitives;
my $retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  Oracle::hlpr::prim_long->_type() );
return $retNew;
}

sub getchecking {
my $self = shift; my $jClass;
use Oracle::hlpr::Primitives;
my $nameJava = Oracle::hlpr::Marshall->marshal( $_[0],
  Oracle::hlpr::prim_string->_type() );
my $ret = $self->{javaObj}->getchecking("$String", $nameJava,
  "Lfinance/checkingaccount;");
return Oracle::hlpr::Excp->throw($ret)
        if (Oracle::hlpr::Excp->isexcp($ret));
use finance::checkingaccount;
my $retNew = Oracle::hlpr::Marshall->unmarshal( $ret,
  finance::checkingaccount->_type() );
return $retNew;
}

1;
```

finance/checkingaccount.pm

```
package finance::checkingaccount;

use finance::account::trans_type;
use finance::account::trans_t;
use finance::account::transaction;

use Oracle::Java::VM qw(:TYPES);
use Oracle::Java::ClassLdr;
use Oracle::Java::Object;
use CORBA::Object;
use Oracle::hlpr::Marshall;
use Oracle::hlpr::Excp;
use CORBA::TypeCode;
use Oracle::hlpr::Primitives;

use finance::account;

@ISA = qw (  finance::account );

sub _type {
if(!defined($finance::checkingaccount::intf_tc)) {
    my $jClass = Oracle::Java::ClassLdr->loadClass (
      "finance/checkingaccountHelper");
    my $id = $jClass->id("$String");
    my $name = "finance::checkingaccount";

    $finance::checkingaccount::intf_tc = CORBA::TypeCode->create_interface_tc(
      $id, $name);
    $finance::checkingaccount::intf_tc->jClass("finance/checkingaccount");
    $finance::checkingaccount::intf_tc->perlClass("finance::checkingaccount");
    $finance::checkingaccount::intf_tc->JNItype("Lfinance/checkingaccount;");
}

return $finance::checkingaccount::intf_tc;
}

1;
```

finance.pm

```
package finance;

use finance::account;
use finance::checkingaccount;
use finance::bank;

*L = ¥3;
1;
```

outer/inner.pm

```
package outer::inner;


*thing = ¥3;
1;
```

outer.pm

```
package outer;

use outer::inner;

1;
```

第 II 部

Perl カートリッジ

10

Perl カートリッジの概要

Perl は CGI スクリプトを作成するために一般的に使用されるインタプリタ言語です。Perl は強力なテキスト処理機能を備えているため、クライアントからのリクエストの解析や、動的 HTML の生成に理想的です。Perl はインターネット上の多くのサイトからダウンロードできます。ポインタのリストは **http://www.perl.com** から入手可能です。

Perl カートリッジには Oracle Application Server の環境で稼動する Perl インタプリタが付属しています。Perl カートリッジを使用せずに、Oracle Application Server により Perl スクリプトを実行する（すなわち、CGI スクリプトとして実行する）こともできますが、Perl カートリッジの下で Perl スクリプトを実行する方が優れたパフォーマンスを得られます。さらに、Perl カートリッジには Perl インタプリタが内蔵されているため、**perl** 実行モジュールをユーザーのシステムに組み込む必要はありません。

Perl カートリッジ用に作成された Perl スクリプトは、CGI 環境用に作成された Perl スクリプトとは若干異なります。Perl カートリッジによるインタプリタの実行方法が異なるためです。CGI 環境で実行される Perl スクリプトがすでにシステムに存在する場合、Perl カートリッジの下で正しく実行できるように、そのスクリプトを変更する必要があります。

Perl カートリッジは、Perl バージョン 5.004_01 をベースにしています。

注意： Oracle Application Server は、SUID ビットが設定された Perl インタプリタ実行モジュールはサポートしていません。

内容

Perl カートリッジのパフォーマンス改善方法

次のような理由で、Perl スクリプトは CGI 環境で実行するよりも Perl カートリッジの下で実行したほうが実行速度が速くなります。

- Perl カートリッジにより、Perl インタプリタが継続的に維持されます。

  これにより、サーバーが Perl の CGI スクリプトを実行するリクエストを受け取るたびに新規のインタプリタの割当ておよび作成を行うオーバーヘッドをなくすことができます。インタプリタはメモリーにいったんロードされると、各リクエストを処理した後も稼動し続けます。

- Perl カートリッジは、Perl スクリプトをプリコンパイルしてキャッシュに入れます。

  Perl カートリッジがリクエストを受け取ったとき、Perl インタプリタはコンパイル済みのスクリプトを実行する準備ができています。スクリプトが前回コンパイルされた後変更されていない限り、スクリプトをコンパイルする必要はありません。スクリプトが変更された場合、Perl カートリッジはこれを検出し、スクリプトの新規バージョンをコンパイルしてキャッシュに入れます。

付属ファイル

Oracle Application Server には、Perl カートリッジの他に、Perl のバイナリ、ソースおよび man ページが付属しています。バイナリや man ページは Oracle Application Server のインストール時に一緒にインストールされますが、Perl ソースはインストールされません。ソース・ファイル（圧縮形式）は、CD から使用可能です。

表 10-1 付属ファイル

ディレクトリ	説明
$ORAWEB_HOME/../cartx/common/perl	Perl カートリッジの最上位ディレクトリ
$ORAWEB_HOME/../cartx/common/perl/lib	Perl カートリッジのライブラリ・ファイルとランタイム・ファイル
$ORAWEB_HOME/../cartx/common/perl/bin	Perl バイナリ
$ORAWEB_HOME/../cartx/common/perl/man	Perl の man ページ（UNIX のみ）
$ORAWEB_HOME/../cartx/common/perl/lib/Pod/html/pod	HTML 形式の Perl ヘルプ・ページ（Windows NT のみ）。トップ・ページは **perl.html** です。
$ORAWEB_HOME/../cartx/common/perl/src	Perl ソース（圧縮形式）。CD からインストールした場合、このディレクトリはインストールされません。

$ORAWEB_HOME/../cartx/common/perl をメインの Perl として使用

Oracle Application Server 付属の Perl をメインの Perl として使用し、Perl スクリプトを Oracle Application Server のコンテキスト外で実行する際にもこれを使用できます。たとえば、Perl スクリプトをシェルから実行する際にも使用できます。

Perl の実行モジュールを使用するには、次のステップを実行します。

1. **$ORAWEB_HOME/../cartx/common/perl/bin** をパスに追加します。

   C シェルを使用する場合 :

   ```
   % set path = ($ORAWEB_HOME/../cartx/common/perl/bin $path)
   ```

   Bourne シェルまたは Korn シェルを使用する場合 :

   ```
   $ PATH=$ORAWEB_HOME/../cartx/common/perl/bin:$PATH; export PATH
   ```

2. PERL5LIB 環境変数を設定します。

   C シェルを使用する場合 :

   ```
   % setenv PERL5LIB $ORAWEB_HOME/../cartx/common/perl/lib/sun4-solaris/
   5.00401:$ORAWEB_HOME/../cartx/common/perl/lib:$ORAWEB_HOME/../cartx/common/perl/
   lib/site_perl/sun4-solaris:$ORAWEB_HOME/../cartx/common/perl/lib/site_perl
   ```

   Bourne シェルまたは Korn シェルを使用する場合 :

   ```
   $ PERL5LIB=$ORAWEB_HOME/../cartx/common/perl/lib/sun4-solaris/
   5.00401:$ORAWEB_HOME/../cartx/common/perl/lib:$ORAWEB_HOME/../cartx/common/perl/
   lib/site_perl/sun4-solaris:$ORAWEB_HOME/../cartx/common/perl/lib/site_perl;
   export PERL5LIB
   ```

Perl の man ページを使用するには、MANPATH 環境変数に **$ORAWEB_HOME/../cartx/common/perl/man** を追加します。MANPATH がすでに設定されている場合は、次のようにします。

C シェルを使用する場合 :

```
%  setenv MANPATH ${MANPATH}:$ORAWEB_HOME/../cartx/common/perl/man
```

Bourne シェルまたは Korn シェルを使用する場合 :

```
$  MANPATH=$MANPATH:$ORAWEB_HOME/../cartx/common/perl/man; export MANPATH
```

注意： 現在、NT 環境では一部の環境変数（CLASSPATH、JAVA_HOME など）の拡張時の長さは 512 バイトに制限されています。Oracle Application Server カートリッジや JCO オブジェクトの中には、環境変数が拡張されるものがあります。したがって、環境変数の長さが必ず 250-300 文字以内になるようにしてください。

標準版の Perl との相違点

Oracle Application Server の Perl カートリッジに付属している Perl は、標準の Perl バージョン 5.004_01 です。インタプリタは UNIX の共有オブジェクト **libperlctx.so**、および NT の共有ライブラリ **perlnt40.dll** として作成されています。Perl と LiveHTML カートリッジは、実行時に共有オブジェクトまたはライブラリにリンクします。

Perl インタプリタを、Perl カートリッジまたは LiveHTML カートリッジと静的にリンクするかわりに、共有オブジェクトまたは動的ライブラリとして使用することにより、インタプリタのアップグレードが可能です。また、このように設計することにより、Perl カートリッジまたは LiveHTML カートリッジから、同じサイトにインストールされている互換性のある他の Perl を使用可能になります。

Perl カートリッジによるスクリプトの実行は、標準 Perl インタプリタによるスクリプトの実行とは次の点で異なります（LiveHTML にも適用されます）。

- 標準 I/O は WRB のクライアント I/O、つまり、クライアントのブラウザにリダイレクトされます。
- STDERR は WRB の Logger にリダイレクトされます。
- Perl インタプリタがシステム環境変数を要求すると、常に、追加の CGI 環境変数が返されます。
- `fork` コールはサポートされていません。かわりに `system` コールを使用します。`system` コールにより Perl インタプリタのインプリメンテーションが変更され、子プロセスの出力が WRB のクライアント I/O にリダイレクトされます。
- エラー・ロギングのサポート。
- パフォーマンス・ツールのサポート。

標準の Perl の実行モジュールは、**http://www.perl.org** から入手可能です。

11
チュートリアル

この章では、Oracle Application Server で Perl アプリケーションを作成する方法を順を追って説明します。このチュートリアルでは、次の作業をステップごとに説明します。

1. Perl スクリプトの作成

2. Perl アプリケーションとコンポーネントの作成

3. リロード

4. Perl スクリプトを実行する HTML ページの作成

サーバーにアプリケーションを追加するには、Oracle Application Server に "admin" ユーザーでログインする必要があります。

1. Perl スクリプトの作成

サンプルの Perl アプリケーションは、いくつかの CGI 環境変数の値を表示する Perl スクリプトを実行します。

ファイルに次の Perl スクリプトを入力し、**showEnv.pl** として保存します。そのファイルを **$ORAWEB_HOME/test** ディレクトリ内に置きます。このディレクトリを作成する許可がない場合、ファイルを別のディレクトリに入れても構いませんが、仮想パス・マッピングを指定する際にディレクトリ名を覚えておいてください。

```
print "Content-type: text/html¥n¥n";

print "<html>¥n";
print "<head>¥n";
print "<title>Some CGI environment variables</title>¥n";
print "</head>¥n";
print "<body bgcolor=white>¥n";

print "<h1>Some CGI environment variables</h1>¥n";

@varsToDisplay = (
```

```
    'HTTP_USER_AGENT',
    'REQUEST_METHOD',
    'PATH_INFO',
    'PATH_TRANSLATED');

print "<dl>¥n";
foreach (@varsToDisplay) {
    print "<dt>$_¥n<dd>$ENV{$_}¥n";
}

print "</dl>¥n";
print "</body></html>¥n";
```

2. Perl アプリケーションとコンポーネントの作成

このステップを実行するには、Oracle Application Server に admin ユーザーでログインする必要があります。

1. ブラウザを起動し、Oracle Application Server の管理ページのトップ・ページを表示します。左フレームの一番上に「website40 サイト」が表示されます。
2. プラス記号（+）をクリックして、そのサイトを表示します。
3. サイト名の横のプラス記号（+）をクリックして、そのサイトのコンポーネントを表示します。「HTTP リスナー」、「Oracle Application Server」および「アプリケーション」が表示されます。
4. 「アプリケーション」をクリックして、右フレームにアプリケーションを表示します。「アプリケーション」の横のプラス記号（+）は、右フレームにアプリケーションを表示するかわりに、左フレームに該当するサイトのアプリケーションをリストするだけなので、クリックしないでください。
5. 右フレームのアプリケーション・ページで、一番上にある緑のプラス・アイコンをクリックします。「アプリケーションの追加」ダイアログ・ボックスが表示されます。
6. 「アプリケーションの追加」ダイアログ・ボックスで次の作業を行います。
   - 「アプリケーション・タイプ」:「Perl」を選択します。
   - 「モードの設定」:「手動」を選択します。これにより、ダイアログ・ボックスを使用して設定データを入力できます。他のオプション「ファイルから」は、アプリケーションの設定データがすでにファイルに入力されていることを前提としています。
   - 「適用」をクリックします。

     「アプリケーションの追加」ダイアログ・ボックスが表示されます。

7. 「アプリケーションの追加」ダイアログ・ボックスで次の作業を行います。
   - 「アプリケーション名」: "showCGIvals" と入力します。この名前は、アプリケーションを識別するために使用されます。
   - 「表示名」: "showCGIvals" と入力します。この名前は、管理用フォームで使用されます。
   - 「アプリケーションのバージョン」: "1.0" と入力します。
   - 「適用」をクリックします。

     「適用」をクリックすると「成功」ダイアログ・ボックスが表示され、カートリッジをアプリケーションに追加できるボタンが表示されます。
8. 「成功」ダイアログ・ボックスで、「このアプリケーションにカートリッジを追加」ボタンをクリックします。「カートリッジの追加」ダイアログ・ボックスが表示されます。
9. 「カートリッジの追加」ダイアログ・ボックスで次の作業を行います。
   - 「カートリッジ名」: "cart1" と入力します。この名前は "showCGIvals" アプリケーション内で Perl カートリッジを識別するために使用されます。
   - 「表示名」: "firstCart" と入力します。この名前は、管理用フォームで使用されます。
   - 「仮想パス」: "/perl/test" と入力します。これは、クライアントがアプリケーションへのアクセスに使用するパスです。
   - 「物理パス」: "%ORAWEB_HOME%/test" と入力します。これは、サーバー上のアプリケーションの物理パスです。
   - 「適用」をクリックします。

3. リロード

Oracle Application Server を再設定したら、サーバーをリロードして新規設定を有効にする必要があります。『Oracle Application Server 管理者ガイド』の「アプリケーションの管理」を参照してください。

また、新しい仮想パスが有効になるように、リスナーを停止して再起動する必要があります。『Oracle Application Server 管理者ガイド』の「アプリケーションの管理」を参照してください。

4. Perl スクリプトを実行する HTML ページの作成

Perl スクリプト **showEnv.pl** を実行するには、ブラウザで次の URL を入力します。

```
http://<host>:<port>/perl/test/showEnv.pl
```

host と *port* には、そのカートリッジを認識するリスナーを指定します。これは、Oracle Application Server 上のどのリスナーでも構いませんが、Node Manager リスナー（デフォルトではポート 8888 上で実行）は含まれません。たとえば、デフォルトでポート 8889 で実行される Administration Utility リスナーを使用できます。

ただし、HTML ページからプロシージャを実行する方が一般的です。たとえば、次の HTML ページには、前述の URL をコールするリンクが含まれています。

```
    <HTML>
    <HEAD>
    <title>CGI Environment Variables</title>
    </HEAD>
    <BODY>
    <H1>CGI Environment Variables</H1>
    <p><a href="http://hal.us.oracle.com:9999/perl/test/showEnv.pl">Show CGI
environment variables</a>
    </BODY>
    </HTML>
```

次の図は、ソース・ページ（**showEnv.pl** スクリプトを実行するリンクが含まれるページ）と、スクリプトによって生成されるページです。

図 11-1 このチュートリアルのソース・ページおよび動的に生成される HTML ページ

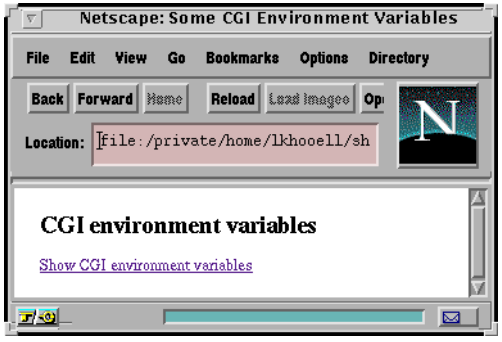
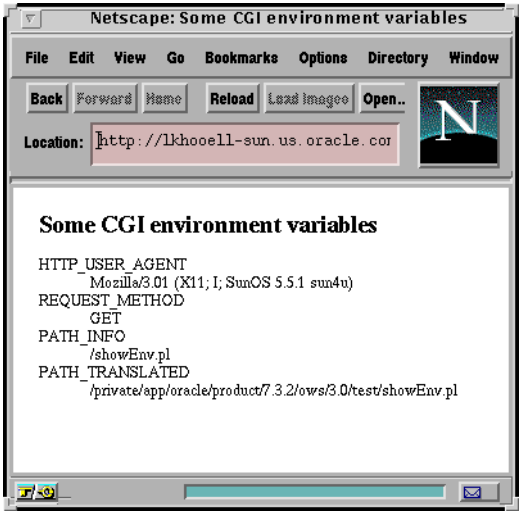


図 11-2 Perl スクリプトによって生成されたページ



12

Perl アプリケーションの追加と実行

Perl アプリケーションを設定するには、Oracle Application Server Manager を使用します。Oracle Application Server Manager は管理用フォームのセットです。これらのフォームで、Perl アプリケーション用の仮想パス、Perl アプリケーションとそのカートリッジのインスタンスの最小値と最大値、仮想パスの保護などの情報を入力します。

内容

Perl アプリケーションの追加

Perl アプリケーションを Oracle Application Server に追加するには、次のステップを実行します。

- アプリケーションを追加する。
- アプリケーションにカートリッジを追加する。

アプリケーションとカートリッジを追加するには、次のステップを実行します。

1. ブラウザを起動し、Oracle Application Server の管理ページのトップ・ページを表示します。
2. サイト名の横の ⊞ をクリックして、そのサイトのコンポーネントを表示します。「Oracle Application Server」、「HTTP リスナー」、「アプリケーション」が表示されます。

3. 「アプリケーション」をクリックして、右フレームにアプリケーションを表示します。「アプリケーション」の横の ⊞ は、右フレームにアプリケーションを表示するかわりに、左フレームにサイトのアプリケーション・リストを表示するため、クリックしないでください。
4. 右フレームのアプリケーション・ページで、  をクリックします。「アプリケーションの追加」ダイアログ・ボックスが表示されます。
5. 「アプリケーションの追加」ダイアログ・ボックスで次の作業を行います。
   - 「アプリケーション・タイプ」:「Perl」を選選択します。
   - 「モードの設定」:「手動」を選択します。これにより、ダイアログ・ボックスを使用して設定データを入力できます。他のオプション「ファイルから」は、アプリケーションの設定データがすでにファイルに入力されていることを前提としています。
   - 「適用」をクリックします。

   「アプリケーションの追加」ダイアログ・ボックスが表示されます。
6. 「アプリケーションの追加」ダイアログ・ボックスで次の作業を行います。
   - 「アプリケーション名」: 設定ファイルに設定するアプリケーション名を入力します。
   - 「表示名」: 管理用フォームで使用される名前を入力します。
   - 「アプリケーションのバージョン」: アプリケーションのバージョンを入力します。
   - 「適用」をクリックします。

   「成功」ダイアログ・ボックスが表示されます。
7. 「成功」ダイアログ・ボックスで、「このアプリケーションにカートリッジを追加」ボタンをクリックします。「カートリッジの追加」ダイアログ・ボックスが表示されます。
8. 「カートリッジの追加」ダイアログ・ボックスで次の作業を行います。
   - 「カートリッジ名」: 設定ファイルに設定するカートリッジの名前を入力します。
   - 「表示名」: 管理用フォームで使用される名前を入力します。
   - 「仮想パス」: Perl カートリッジを実行する場合に URL で指定する形式と同じ形式で、Perl カートリッジのパスを入力します。このパスは、次に指定する物理パスにマップされます。この後の「「仮想パス」フォーム」の項を参照してください。
   - 「物理パス」: Perl カートリッジ用のファイル（Perl アプリケーション用のファイルを含む）の場所を示す物理ディレクトリ・パスを入力します。上で指定した仮想パスはこの物理パスにマップされます。
9. 「適用」をクリックします。

10. リスナーおよび Oracle Application Server の他のコンポーネントを停止して再起動します。

    詳細は、『Oracle Application Server 管理者ガイド』の「アプリケーションの管理」を参照してください。

> 注意： 追加したアプリケーションをナビゲーション・ツリーに表示するには、[Shift] キーを押しながらブラウザの「リロード」ボタンをクリックします。

既存のアプリケーションへのカートリッジの追加

Perl アプリケーションには、1 つ以上のカートリッジを持たせることが可能です。カートリッジのパラメータに異なる値を設定する必要がある場合、Perl アプリケーションには 2 つ以上のカートリッジが必要です。たとえば、それぞれのカートリッジに異なる初期化スクリプトを指定できます。

Perl アプリケーションにカートリッジを追加するには、次のステップを実行します。

1. ナビゲーション・ツリーで、カートリッジの追加先の Perl アプリケーションの下の「カートリッジ」を選択します。

図 12-1 既存のアプリケーションへの Perl カートリッジの追加

2.  をクリックして、「カートリッジの追加」ダイアログ・ボックスを表示します。
3. 「カートリッジの追加」ダイアログ・ボックスで次の作業を行います。
   - 「モードの設定」:「手動」を選択します。
   - 「適用」をクリックすると「カートリッジの追加」ダイアログ・ボックスが表示されます。

4. 「カートリッジの追加」ダイアログ・ボックスで次の作業を行います。

   - 「カートリッジ名」: サーバーがアプリケーションの Perl カートリッジを識別するために使用する名前を入力します。
   - 「表示名」: 管理用フォームで使用される名前を入力します。
   - 「仮想パス」: Perl カートリッジを実行する場合に URL で指定する形式と同じ形式で、Perl カートリッジのパスを入力します。このパスは、次に指定する物理パスにマップされます。この後の「「仮想パス」フォーム」の項を参照してください。
   - 「物理パス」: Perl カートリッジ用のファイル（Perl アプリケーション用のファイルを含む）の場所を示す物理ディレクトリ・パスを入力します。上で指定した仮想パスはこの物理パスにマップされます。

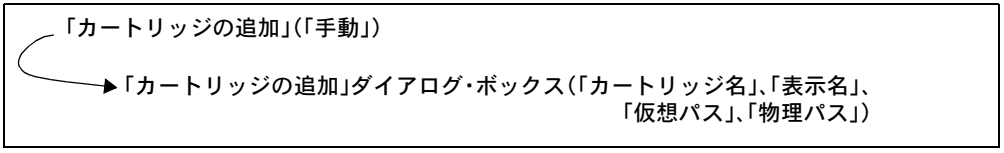
     注意： セキュリティの理由で、最後に ".." が付く物理パスは指定できません。ただし、物理パスの設定で上位のディレクトリ・レベルを示すために、".." を使用することは可能です。たとえば、"/routines/../libraries/" などです。

   - 「適用」をクリックします。

     注意： 追加した新規カートリッジをナビゲーション・ツリーに表示するには、[Shift] キーを押しながらブラウザの「リロード」ボタンをクリックします。

次の図は、入力が完了したダイアログ・ボックスのサマリーです。ダイアログ・ボックスのフィールドはカッコ内にリストされています。

図 12-2 Perl カートリッジを追加するダイアログ



Perl アプリケーションの設定

設定用フォームは、「アプリケーション」の「設定」と「カートリッジ」の「設定」の 2 つのセクションに分かれています。「アプリケーション」の「設定」セクションのフォームには、アプリケーション全体に適用されるパラメータが入っています。一方、「カートリッジ」の「設定」セクションのフォームには、特定のカートリッジにのみ適用されるパラメータが入っています。

「Perl アプリケーションの設定」フォームを使用して、Perl カートリッジ固有のパラメータを定義できます（図 12-3 を参照）。表 12-1 に、このフォームのパラメータを示します。

図 12-3 「Perl アプリケーションの設定」フォーム

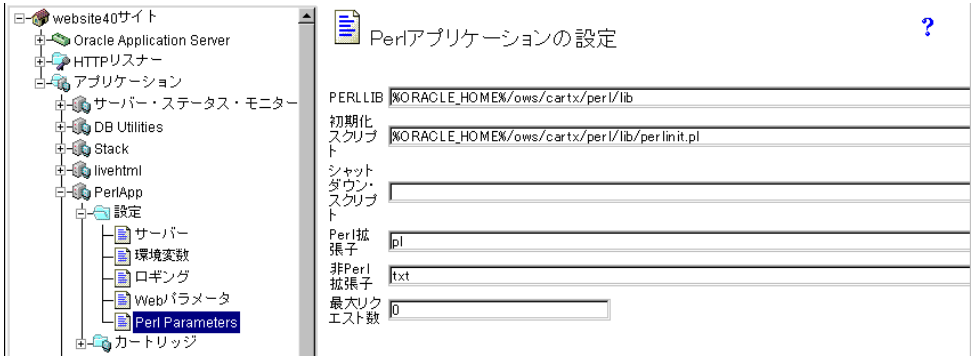


表 12-1 Perl アプリケーションのパラメータ

名前	値
PERLLIB	Perl ライブラリのパス。 デフォルト : %ORAWEB_HOME%/.../cartx/perl/lib
初期化スクリプト	Perl カートリッジのインスタンス起動時に実行されるスクリプト。 デフォルト : %ORAWEB_HOME%/.../cartx/perl/lib/perlinit.pl
シャットダウン・スクリプト	Perl カートリッジのインスタンスをシャットダウンするために実行されるスクリプト。
Perl 拡張子	Perl カートリッジによって処理される有効な Perl スクリプトが含まれているファイルの拡張子。
非 Perl 拡張子	Perl カートリッジによって処理されないファイルの拡張子。 通常、これらのファイルには Perl コードは含まれません。 非 Perl ファイル拡張子を指定すると、Perl ファイルと非 Perl ファイルを同じディレクトリに格納できます。
最大リクエスト数	カートリッジ・サーバーが終了するまでに処理可能なリクエストの数。

カートリッジ・インスタンスが処理するリクエスト数

一部の AUTOLOAD サブルーチンでは、カートリッジが追加のリクエストを受け取るにつれて、Perl カートリッジのシンボル・テーブルが大きくなってしまうことがあります。これを制限するには、「最大リクエスト数」に小さい値を設定してください。

「カートリッジ」の「設定」

Perl カートリッジの場合、「カートリッジ」の「設定」セクションには「仮想パス」フォームと「チューニング」フォームの 2 つのフォームが含まれます。

「仮想パス」フォーム

「仮想パス」フォームを使用して、Perl カートリッジの仮想パスを指定できます。ユーザーは URL にこの仮想パスを指定してカートリッジを実行します。仮想パスは、そのアプリケーションの「Web の設定」ページにリストされているすべてのリスナーから使用可能です。

たとえば、ある Perl カートリッジに対して **/myApp** という仮想パスを指定した場合、ユーザーは **/myApp/***file* と入力することによりそのカートリッジを実行できます。*file* には **/myApp** 仮想パスに関連付けられた物理パスに存在する Perl スクリプトを指定します。

「チューニング」フォーム

「チューニング」フォームでは、カートリッジの優先順位および起動インスタンスを指定できます。

Perl カートリッジの実行

Perl カートリッジの下で Perl スクリプトを実行するには、次の形式の URL を使用する必要があります。

`http://`*`host_and_domain_name`*`[:`*`port`*`]/`*`virtual_path`*`/`*`script_name`*`[?`*`query_string`*`]`

詳細は次のとおりです。

- *host_and_domain_name* には、Oracle Application Server が稼動しているドメインとマシンを指定します。
- *port* には、Oracle Application Server がリスニングしているポートを指定します。指定しない場合、ポート 80 が使用されます。
- *virtual_path* には、Perl カートリッジにマップされている仮想パスを指定します。
- *script_name* には、Perl スクリプトが入っているファイルを指定します。規則により、Perl スクリプトには拡張子 ".pl" が付きます。また、Node Manager の Perl アプリケーション内の Perl パラメータ・ブランチで、Perl スクリプトに他の拡張子を指定することも可能です。
- *query_string* には、スクリプトのパラメータを指定します。

たとえば、ブラウザが次の URL を送信すると

http://www.acme.com:9000/perl/myScript.pl

www.acme.com で稼動し、ポート **9000** でリスニングしているサーバーがリクエストを処理します。リスナーはリクエストを受け取ると、それを WRB に渡します。これは仮想ディレクトリ `/perl` が、Perl カートリッジをコールするように設定されているためです。次に、Perl カートリッジが **myScript.pl** を実行します。

Perl カートリッジのライフ・サイクル

この項では、Perl カートリッジがリクエストを受け取った際に行う処理について説明します。この項は、Oracle Application Server（WRB）で使用されるコールバック関数の知識があることを前提としています。

Perl カートリッジを使用するために、この項の情報を知っておく必要はありません。ただし、カートリッジのアーキテクチャを知る上でこの情報は便利です。

カートリッジ・サーバーで Perl カートリッジの最初のインスタンスが起動されると、そのインスタンスにより initRuntime コールバック関数が実行されます。initRuntime 関数は Perl インタプリタを構成し、起動スクリプト **$ORAWEB_HOME/../cartx/common/perl/lib/persist.pl** を実行します。次に、この関数は InitScript 設定パラメータで指定された Perl スクリプトを、**persist.pl** スクリプトのコンテキストのサブルーチンとして実行します。InitScript のデフォルト値は、**$ORAWEB_HOME/../cartx/common/perl/lib/perlinit.pl** です。

InitScript によって指定されたスクリプトによって、他の Perl スクリプトで必要となる初期化の作業を実行できます。たとえば、この時点でモジュールをロードしたり、データベースに接続できます。

カートリッジが URL のリクエストを認証する必要がある場合、authorize コールバック関数が実行されます。authorize 関数は、リクエストされたオブジェクトが認証スキーマや制限によって保護されているかどうかを調べます。authorize コールバック関数が成功すると、次に exec コールバック関数がコールされます。exec 関数は次の処理を行います。

- CGI 環境変数の値を取得する。
- 実行する Perl スクリプトを決定する。
- プロシージャのパラメータを決定する。

カートリッジは、URL に指定された Perl スクリプトをコンパイルし、Perl スクリプト名をベースにした名前のパッケージにサブルーチンとして保管します。これによって、スクリプトの変数やサブルーチンをパッケージに配置できます。

リクエストされた Perl スクリプトは、起動スクリプト **persist.pl** のコンテキストのサブルーチンとして実行されます。その後、同じ Perl スクリプトに対するリクエストを受け取った際に、スクリプトは（変更されていない限り）再コンパイルされません。かわりに、Perl スクリプトに対応するパッケージ名が生成され、そのパッケージ内の一意に識別されるサブルーチンがコールされます。

shutdown コールバック関数は、カートリッジ・インスタンスをシャットダウンするために Oracle Application Server によって自動的にコールされます。shutdown 関数は、インスタンスをシャットダウンする前に ShutScript 設定パラメータで指定された Perl スクリプトを実行します。このスクリプトによって、InitScript 設定パラメータで指定されたスクリプトが行なった初期化をすべて解除できます。ShutScript のデフォルト値は、**$ORAWEB_HOME/../cartx/common/perl/lib/perlshut.pl** です。shutdown コールバック関数は、スクリプトを実行した後に Perl インタプリタの割当てを解除し、Perl インタプリタを破棄します。

13

Perl スクリプトの作成

Perl スクリプトを Perl カートリッジの下で正しく実行するためには、いくつかのルールを守る必要があります。これらのルールに合うように、既存の Perl の CGI スクリプトを変更する必要がある場合もあります。

内容

カスタマイズされた cgi-lib.pl ライブラリ

Perl スクリプトで **cgi-lib.pl**（最新情報は **http://cgi-lib.stanford.edu/cgi-lib/** を参照）を使用する場合、Perl カートリッジ用にカスタマイズされたバージョンのライブラリを使用するように、スクリプトを変更する必要があります。Perl カートリッジは Perl インタプリタを継続的に実行するため、変更されていないバージョンの **cgi-lib.pl** は Perl カートリッジでは使用できません。変更されたバージョンの **cgi-lib.pl** は **$ORAWEB_HOME/sample/perl/mycgi-lib.pl** です。

Perl スクリプトで、次のように記述するかわりに

```
require "./cgi-lib.pl";
```

次の行を使用します。

```
$ORACLE_HOME = $ENV{'ORACLE_HOME'};
$ORAWEB_HOME = "$ORACLE_HOME/ows/4.0";
require "$ORAWEB_HOME/../cartx/common/perl/sample/mycgi-lib.pl";
```

変数のスコープ

Perl スクリプトを Perl カートリッジの下で実行する際、名前領域と変数のスコープに注意してください。従来の CGI スクリプトでは、変数を宣言して使用します。スクリプトはリクエストのたびに再起動され、かつリエントラントではないため、変数の定義を解除する必要はありません。

Perl カートリッジの場合、グローバル変数は複数のコールに渡って維持されます。スクリプトを実行して終了したときの変数の値は、次回、そのスクリプトを実行するときの初期値となります。このため、次の例のように、出力に一貫性がない場合があります。

```
1  print "Content-type: text/plain¥n¥n";
2  @question = (where, are, you, staying);
3  print "@question¥n";
4  $" = "¥n";
5  @answer = (all, on, separate, lines);
6  print "@answer¥n";
```

そのカートリッジで初めて実行された際には、次のように出力されます。

```
where are you staying
all
on
separate
lines
```

2 回目以降に実行された際には、次のように出力されます。

```
where
are
you
staying
all
on
separate
lines
```

これは、スクリプトの第 4 行で Perl の特殊変数 $" が "¥n" に変更されていて、スクリプトの終了前に元の値にリセットされていないためです。Perl カートリッジでは、特殊変数の初期値は ""（ブランク）です。

この問題の解決策の 1 つは、スクリプトの最後に次の行を追加して、変数の値をリセットすることです。

```
$" = " ";
```

スクリプトの実行がグローバル変数の値に依存する場合は、これらの変数が元の値にリセットされているかどうか確認してください。

前述の問題を避けるには、次のようにします。

- 変数のスコープを必要な範囲のみに限定する。変数を使用後に（my() を使用してその変数のスコープを設定することによって）無効にするか、スクリプトを使用して元の値にリセットするようにしてください。
- グローバル変数を減らし、可能な場合は変数をローカルにする。Perl のグローバル変数を変更する必要がある場合は、その変数をローカルにして、その変更が変数のローカル・インスタンスにのみ反映されるようにしてください。その結果、変更した変数の値は、スクリプトのその実行時のみ有効になります。

上の例の別の解決方法は、$" 変数をローカルにすることです。

```
1  print "Content-type: text/plain¥n¥n";
2  @question = (where, are, you, staying);
3  print "@question¥n";
4  local($");
5  $" = "¥n";
6  @answer = (all, on, separate, lines);
7  print "@answer¥n";
```

第 4 行で Perl の特殊変数 $" をローカル変数にしています。このスクリプトがパッケージのサブルーチンとして実行される場合、ローカル変数 $" は、スクリプトのその実行時のみ有効です。

名前領域の競合

Perl カートリッジは、レスポンス時間を短縮するためにコンパイル済みの Perl スクリプトをキャッシュに入れます。複数の Perl スクリプトをキャッシングした場合、正しく処理しないと名前領域の競合が起こる可能性があります。名前領域の競合を避けるために、Perl カートリッジは Perl スクリプトのファイル名を一意のパッケージ名に変換し、その後 eval を使用してコードをパッケージにコンパイルします。これで、そのスクリプトは一意のパッケージ名のサブルーチンとしてコンパイル済みの形式となり、Perl カートリッジから使用可能になります。スクリプトがリクエストを受け取ると、カートリッジはファイル名をパッケージ名に変換し、サブルーチン・ハンドラを実行します。

前述のメカニズムにより名前領域の競合を防ぐことができますが、スクリプトのデフォルト・パッケージ名は "main" ではなくなるので注意してください。スクリプトのデフォルト・パッケージ名は、カートリッジによって生成される名前になります。

次の例では、異なるパッケージ名がどのように Perl スクリプトに影響を与えるかを示します。

Perl にはライブラリ・ファイルが用意されています。これは Perl スクリプトで、ユーティリティ機能を備えています。たとえば、**bigint.pl** は Perl に任意のサイズの整数の数理サブルーチンを提供しています。このライブラリでは、サブルーチンの多くがパッケージ "main" の名前領域に定義されています。次はその例です。

```
# normalize string form of number.  Strip leading zeros.  Strip any
#   white space and add a sign, if missing.
# Strings that are not numbers result the value 'NaN'.
sub main'bnorm { #(num_str) return num_str
    local($_) = @_;
    s/¥s+//g;                           # strip white space
    if (s/^([+-]?)0*(¥d+)$/$1$2/) {     # test if number
        substr($_,$[,0) = '+' unless $1; # Add missing sign
        s/^-0/+0/;
        $_;
    } else {
        'NaN';
    }
}
```

従来の Perl の CGI スクリプトでは、次のようにしてこのライブラリを使用できます。

```
1  # namespace.pl
2  require "bigint.pl";
3  print "Content-type: text/plain¥n¥n";
4  $one =  &bnorm(  456);
5  print "one = $one¥n";
```

第 4 行では、パッケージ名を指定せずに `bnorm` サブルーチンをコールできます。これは、標準的なスクリプトのデフォルトのパッケージ名が "main" なので、"main" パッケージの名前領域にある `bnorm` サブルーチンを使用できるためです。しかし、Perl カートリッジでは、スクリプトはパッケージにコンパイルされ、その名前はファイル名によって決まります。`eval` 文は、すべて（`require` が `eval` をコールすることに注意）このパッケージの名前領域内で評価されます。前述の例では、**bigint.pl** はコンパイルされてスクリプトのパッケージに保管されますが、サブルーチンは完全修飾のサブルーチン名（たとえば、main'bnorm）で "main" パッケージに明示的に保管されます。

Perl カートリッジの下でこの例を実行するには、bnorm を main'bnorm としてコールするようにスクリプトを変更する必要があります。

```
1  # namespace.pl
2  require "bigint.pl";
3  print "Content-type: text/plain¥n¥n";
4  $one =  &main'bnorm(  456);
5  print "one = $one¥n";
```

デフォルト・パッケージの名前は必ずしも "main" とは限りません。現在使用しているパッケージの名前を調べるには、Perl の caller() 関数を実行します。この関数により、Perl カートリッジによって生成されたパッケージ名、ファイル名および現在の行番号が入ったリストが返されます。

#! 行が不要

一般的に、Perl の CGI スクリプトの第 1 行で Perl インタプリタの場所をシステムに知らせます。この行は "#!" で始まり、次のようになります。

```
#!/usr/bin/perl
```

Perl カートリッジの下で実行される Perl スクリプトの場合、この行は必要ありません。Perl カートリッジは組込みの Perl インタプリタを使用するためです。

システム・リソース

Perl スクリプトによって取得したシステム・リソースは、そのスクリプトが終了する前に解除する必要があります。そうしないと、Perl カートリッジの永続的な Perl インタプリタがリソースを要求してシステムの限界に達することがあります。

従来の Perl の CGI スクリプトでは、ファイルを開いてファイル操作を行った場合、スクリプトの終了前にファイルを閉じる必要はありません。この場合、Perl インタプリタの終了時にリソースが返されるため問題は起こりません。しかし、Perl カートリッジの環境では、スクリプトの実行が終了した後もファイルは開いたままになります。したがって、ユーザーがスクリプトで明示的にファイルを閉じる必要があります。

DBI と DBD::Oracle モジュール

DBI と DBD::Oracle モジュールを使用すると、Oracle7 と Oracle8 の両方のデータベースにアクセスできます。DBI（データベース・インタフェース API）モジュールは、データベースの種類に依存しない、一貫性のある Perl スクリプトへのデータベース・アクセス API を提供しています。DBD::Oracle モジュールを使用すると、DBI API によって Oracle7 および Oracle8 データベースにアクセスできます。Oracle Application Server 付属の DBD::Oracle モジュール（バージョン 0.44）は、Oracle 8 データベースのクライアント・ライブラリをベースにしています。

モジュールのプリロード - 永続的なデータベース接続

作成した Perl スクリプトには、スクリプトのリクエストのたびに繰り返し実行する必要がない命令が含まれていることがあります。このような命令を 1 回だけ実行し、各 Perl スクリプトのリクエスト時には必要な部分のみ実行するようにすると、パフォーマンスが向上します。たとえば、Perl スクリプトから DBI と DBD::Oracle モジュールを使用して Oracle データベースにアクセスすると、そのスクリプトへのリクエストが発生するたびにモジュールをロードしてログインするのではなく、Perl カートリッジのインスタンスの起動時にモジュールをプリロードしてデータベースにログインできます。スクリプトでカーソルをオープンし、そのカーソルを使用して結果をフェッチして返します。このようにモジュールをプリロードすることにより、複数のリクエストに渡ってデータベース接続を継続することができます。

DBI と DBD::Oracle モジュールは、**$ORAWEB_HOME/../cartx/common/perl/lib/site_perl** ディレクトリに入っています。

モジュールをプリロードして初期タスクを実行するには、**$ORAWEB_HOME/../cartx/common/perl/lib/perlinit.pl** ファイルを編集します。このファイルは、カートリッジ・インスタンス起動時に一度だけ実行されます。デフォルトでは、このファイルには実行可能な文は含まれていません。このファイルは InitScript パラメータで指定され、その値は「Perl カートリッジの設定」ページで変更できます。

次は、データベースにログインし、問合せを行い、出力を表示する典型的な Perl スクリプトの例です。このスクリプト名を **scott.pl** とします。

```
1   use DBI;
2   print "Content-type: text/plain¥n¥n";
3   $dbh = DBI->connect("", "scott", "tiger", "Oracle") || die $DBI::errstr;
4   $stmt = $dbh->prepare("select * from emp order by empno") || die $DBI::errstr;
5   $rc = $stmt->execute() || die $DBI::errstr;
6   $nfields = $stmt->rows();
7   print "Query will return $nfields fields¥n¥n";
8   while (($empno, $name) = $stmt->fetchrow()) { print "$empno $name¥n"; }
9   warn $DBI::errstr if $DBI::err;
10  die "fetch error: " . $DBI::errstr if $DBI::err;
11  $stmt->finish() || die "can't close cursor";
12  $dbh->disconnect() || die "cant't log off Oracle";
```

第 1 行で DBI モジュールをロードしています。DBI モジュールのロードは、Perl モジュール・スクリプト（**DBI.pm**）と共有オブジェクト（**DBI.so**）の読込みによって行われます。これらのファイルは、Perl スクリプトから Oracle データベースに接続するコールを定義することによって Perl 言語を拡張しています。いったんモジュールがロードされると、共有オブジェクトがカートリッジに動的にリンクされ、Perl スクリプト .pm が読み込まれてコンパイルされ、パッケージの名前領域に格納されます。その後、.pm ファイルで定義されたコールをスクリプトから使用できるようになります。この例では、DBI モジュールの `connect()`、`prepare()`、`execute()`、`rows()`、`fetchrow()`、`finish()`、`disconnect()` メソッドをコールしています。

DBI モジュールをプリロードし、データベースにログインするようにこの例を変更するには、次の行を **perlinit.pl** ファイルに追加します。

```
1   # Contents of perlinit.pl
2   package Scott;
3   use DBI;
4   $dbh = DBI->connect("", "scott", "tiger", "Oracle") || die $DBI::errstr;
```

第 2 行でパッケージ名を Scott と宣言しています。

第 3 行で DBI モジュールをロードしています。共有オブジェクト **DBI.so** は、カートリッジ・プロセスのアドレス空間に動的にリンクされ、.pm ファイルは、コンパイルされて Scott パッケージの名前領域に格納されます。

第 4 行では DBI モジュールの `connect()` メソッドをコールしています。このメソッドは、TWO_TASK 環境変数で指定された Oracle データベースに接続し、ログインします。また、`connect()` コールの最初のパラメータで指定すると、リモート・データベースに接続することも可能です。接続のコンテキストは `$dbh` 変数に格納されます。スクリプトからこのコンテキストにアクセスするには、次のようにスクリプトが同じパッケージの名前領域内（Scott 名前領域内）にあることを宣言する必要があります。

scott.pl の内容は次のとおりです。

```
1   # Contents of scott.pl
2   package Scott;
3   print "Content-type: text/plain¥n¥n";
4   $stmt = $dbh->prepare("select * from emp order by empno") || die $DBI::errstr;
5   $rc = $stmt->execute() || die $DBI::errstr;
6   $nfields = $stmt->rows();
7   print "Query will return $nfields fields¥n¥n";
8   while (($empno, $name) = $stmt->fetchrow()) { print "$empno $name¥n"; }
9   warn $DBI::errstr if $DBI::err;
10  die "fetch error: " . $DBI::errstr if $DBI::err;
11  $stmt->finish() || die "can't close cursor";
```

スクリプトの 2 つの部分が確実に同じパッケージの名前領域にコンパイルされるようにする必要があります。そのため、**perlinit.pl** スクリプトとユーザー・スクリプトの始めの部分に "`package` *`PackageName`*" を宣言します。

変更後の **scott.pl** では `disconnect()` をコールしていないことに注意してください。これは、データベース接続とログインがカートリッジの起動時にのみ行われるためです。ログアウトすると、それ以降 **scott.pl** は実行できません。そのカートリッジ・プロセスを終了し、**perlinit.pl** スクリプトを実行してデータベース接続を再度確立するような、別のインスタンスを起動する必要があります。`disconnect()` は、ShutScript 設定パラメータで指定された Perl スクリプト内で実行されます。デフォルト値は **$ORAWEB_HOME/../cartx/common/perl/lib/perlshut.pl** です。この例を完成させるには、次の行を含むように **perlshut.pl** を変更します。

```
1   # Contents of perlshut.pl
2   package Scott;
3   $dbh->disconnect() || die "cant't log off Oracle";
```

別のスクリプト（たとえば、**kane.pl**）用に別の継続的なデータベース接続を確立するには、**perlinit.pl** に入れる **kane.pl** のコードを、一意のパッケージ名（たとえば、"Kane"）を付けて挿入します。**kane.pl** スクリプトでも、**perlinit.pl** で初期化された変数やサブルーチンを使用する前に "package Kane" を宣言する必要があります。

Perl スクリプトのテスト

Perl カートリッジには **persistperl** というツールがあり、Perl カートリッジの下で実行される Perl スクリプトをシミュレートします。このツールを使用して、作成した Perl スクリプトがカートリッジの下で正しく実行できるかどうかをチェックします。

persistperl ツールを使用するには、次の作業を行います。

1. パス変数に **$ORACLE_HOME/ows/cartx/perl/bin** ディレクトリを追加します。このディレクトリには **persistperl** バイナリが入っています。
2. 現行ディレクトリを **$ORAWEB_HOME/../cartx/common/perl/lib** に変更します。
3. **persistperl** をシェルから起動します。構文は次のとおりです。

   `persistperl` *script_name* [*number_of_executions*]

script_name には実行する Perl スクリプトの名前を指定し、*number_of_executions* には実行する回数を指定します。指定を省略すると、*number_of_executions* のデフォルトは 100 になります。**persistperl** はスクリプトの出力を標準出力に表示します。ファイルにリダイレクトすることもできます。出力に矛盾がないかチェックします。さらに、この出力は、複数回実行した後でリソース獲得のためにシステムの限界に達したかどうかも示します。

Perl モジュール

この項では、Perl カートリッジ付属の Perl モジュールについて説明します。Perl モジュールは、ライブラリ・ファイルで定義された再使用可能なパッケージです。これらはクラスとして機能し、そのメソッドはモジュールをインポートしたどの Perl スクリプトからでも使用できます。一部の Perl モジュールは標準の Perl の配布物に含まれています（たとえば、DynaLoader、AutoLoader、POSIX）。

これらのパッケージの詳細情報は、**$ORAWEB_HOME/../cartx/common/perl/man** ディレクトリにインストールされているパッケージの man ページを参照してください。

これらのパッケージの最新バージョンについては、**http://www.perl.com/CPAN/** を参照してください。（CPAN は、Comprehensive Perl Archive Network の略称です。）

DBI（バージョン 0.79）

このモジュールには、Perl 言語用のデータベース・アクセス API が記述されています。DBI には、実際に使用されるデータベースに依存しない、一貫性のあるデータベース・インタフェースを提供する関数、変数および規則が定義されています。

DBI はドライバとアプリケーション間のインタフェースにすぎません。データベース・アクセスに必要な実際の作業はドライバが行います。

Oracle データベースへのアクセスに必要なデータベース・ドライバ（DBD::Oracle）は、Perl カートリッジに付属しています。

DBD::Oracle（バージョン 0.44）

このモジュールは、Oracle データベース用のデータベース・ドライバで構成されています。DBD::Oracle モジュールと DBI の詳細は、**http://www.hermetica.com/technologia/DBI/** を参照してください。

DBD::Oracle モジュールを使用してデータベースにアクセスする場合、モジュールをプリロードして永続的なデータベース接続を維持することによってパフォーマンスを改善できます。詳細は、13-6 ページの「モジュールのプリロード - 永続的なデータベース接続」を参照してください。

LWP または libwww-perl（バージョン 5.08）

これは Perl モジュールの集まりで、Web クライアントの作成、HTML の解析、いろいろなサーバー（メール・サーバー、HTTP サーバー、NNTP サーバーなど）との通信に使用可能なクラスが用意されています。LWP には、次の 3 つのクラスがあります。

- LWP::Request

  このクラスはクライアントによって作成され、リクエストおよび関連するリクエスト・データをすべて指定します。Web からドキュメントを受け取るというリクエストの場合、そのリクエスト・クラスはドキュメントの URL とドキュメント名を使用して作成されます。

- LWP::Response

  このクラスは、サーバーによってリクエストへのレスポンスとして返されます。たとえば、リクエストがドキュメントの要求であればドキュメントが返され、メールの送信であればメールが送信されたという確認が返されます。

- LWP::UserAgent

  このクラスは、リクエストとレスポンスとの間のインタフェースを提供しています。クライアントはリクエストを作成して、それを UserAgent に渡します。UserAgent は基礎を形成する低レベル通信および処理を行い、リクエストをサーバーに渡します。次に、リクエストの結果が入ったレスポンスをクライアントに返します。

CGI（バージョン 2.36）

このモジュールは、HTML フォームをすばやく作成し、その内容を解析できるクラスを提供しています。また、CGI スクリプトに渡される問合せ文字列の解析および変換を行うインタフェースも提供しています。このモジュールでは、前回の問合せの値を使用してフォームを初期化できます。つまり、実行のたびにフォームの状態が保存されます。

MD5（バージョン 1.7）

このモジュールによって、Perl プログラムから RSA Data Security MD5 Message Digest のアルゴリズムが使用できます。このモジュールは、libwww-perl のインプリメンテーションによって使用されます。

IO（バージョン 1.15）

このモジュールには、ファイル、パイプおよびソケット I/O の操作を扱うサブモジュールが含まれています。このモジュールに含まれているクラスは次のとおりです。

- IO::Handle

  これは他のすべての IO クラスの基本クラスです。通常、このクラスは直接は使用されません。他の IO クラスがインプリメンテーションの際にこのクラスを継承します。

- IO::Seekable

  このクラスは他の IO クラスにシーク・ベースのメソッドを提供します。提供されるメソッドには、seek、tell および clearerr が含まれます。

- IO::File

  このクラスは、IO::Handle と IO::Seekable を継承します。これには、open、getpos、setpos など、ファイル処理固有のメソッドが含まれています。

- IO::Pipe

  このクラスは、パイプ・ベースのプロセス間通信を作成するインタフェースを提供します。

- IO::Socket

  このクラスは、ソケット通信のためのインタフェースを提供します。ソケットを作成して使用するためのメソッドが含まれています。提供されるメソッドには socket、socketpair、accept、send、recv が含まれます。

Net（バージョン 1.0502）

このモジュールはクラスの集まりで、クライアント側にいろいろなプロトコルをインプリメントするための一貫性のある API がユーザーに提供されます。Net モジュールのサブモジュールは、次のプロトコルをインプリメントしています。

- Net::FTP - ファイル転送プロトコル
- Net::SMTP - Simple Mail Transfer Protocol
- Net::Time - Time および Daytime プロトコル
- Net::NNTP - Network News Transfer Protocol
- Net::POP3 - Post Office Protocol

Net モジュールのインストールを完了するには、設定スクリプトを実行する必要があります。

1. Perl カートリッジに付属している **perl** 実行モジュールが含まれるようにパスを設定します。

   C シェルを使用する場合 :

   ```
   prompt> set path = ($ORAWEB_HOME/../cartx/common/perl/bin $path)
   ```

   Bourne シェルまたは Korn シェルを使用する場合 :

   ```
   prompt> PATH=$ORAWEB_HOME/../cartx/common/perl/bin:$PATH; export $PATH
   ```

2. PERLLIB 環境変数を設定します。

   C シェルを使用する場合 :

   ```
   prompt> setenv PERLLIB $ORAWEB_HOME/../cartx/common/perl/lib:
       $ORAWEB_HOME/../cartx/common/perl/lib/platform/version:
       $ORAWEB_HOME/../cartx/common/perl/lib/site_perl:
       $ORAWEB_HOME/../cartx/common/perl/lib/site_perl/platform/
                                                       version
   ```

   Bourne シェルまたは Korn シェルを使用する場合 :

   ```
   prompt> PERLLIB=$ORAWEB_HOME/../cartx/common/perl/lib:
       $ORAWEB_HOME/../cartx/common/perl/lib/platform/version:
       $ORAWEB_HOME/../cartx/common/perl/lib/site_perl:
       $ORAWEB_HOME/../cartx/common/perl/lib/site_perl/platform/
                                                         version;
       export $PERLLIB
   ```

platform と *version* を該当する値に置き換えてください。*platform* ディレクトリは、インストールされる Perl によって変わります。たとえば、Solaris の場合、*platform* は `sun4-solaris` です。*version* ディレクトリは、OS のバージョンに従って付けられたディレクトリ名を参照します。たとえば、Solaris の場合、`5.00401` などです。

3. ディレクトリを **$ORAWEB_HOME/../cartx/common/perl/lib/site_perl/Net** に変更して、スクリプト **configure.pl** を実行します。

```
prompt> cd $ORAWEB_HOME/../cartx/common/perl/lib/site_perl/Net
prompt> perl configure.pl
```

このスクリプトはマシンに関する設定情報のプロンプトを表示し、設定情報を含んだ **Config.pm** という名前の Perl スクリプトを作成します。このスクリプトは、Net モジュール内のすべてのパッケージに必要です。

Data-Dumper（バージョン 2.07）

このモジュールは、Perl のデータ構造体を永続的にするために必要です。このモジュールは、受け取ったスカラーやリファレンス変数のリストを、Perl の構文で出力するメソッドを備えています。戻り値に `eval` を使用して元のデータ構造体に戻すこともできます。このモジュールは、Perl カートリッジに提供されている他のモジュール（たとえば、net や libwww-perl）のインプリメンテーションで使用されます。

次に、このパッケージの簡単な使用例を示します。

```
use Data::Dumper;
$boo = [ 1, [], "abcd", {1 => 'a', 22 => 'b'}, ¥¥"p¥q¥'r"];
print Dumper($boo);          # Pretty Prints the data-structure
$bar = Dumper($boo);         # $bar has the Perl statement which when
                             # eval-ed returns the original data structure.
print Dumper(eval($bar));    # Pretty prints $boo.
```

Perl の拡張モジュールの開発

Perl カートリッジには Perl インタプリタ実行時環境が付属しています。この Perl インタプリタ実行時環境を使用して開発した Perl 拡張モジュールは、Perl カートリッジからアクセスできます。

Perl インタプリタは **$ORAWEB_HOME/../cartx/common/perl** ディレクトリにインストールされます。**$ORAWEB_HOME/../cartx/common/perl/bin** ディレクトリに "perl" 実行モジュールが入っています。拡張モジュールを開発して、**$ORAWEB_HOME/../cartx/common/perl/lib** ディレクトリにインストールできます。

このためには、次の環境変数を設定する必要があります。

- **$ORAWEB_HOME/../cartx/common/perl/bin** ディレクトリの **perl** 実行モジュールを使用できるように、パス変数を設定します。
- PERL5LIB 環境変数を次のように設定します。

```
$ORAWEB_HOME/../cartx/common/perl/lib/sun4-solaris/5.00401:$ORAWEB_HOME/../
cartx/common/perl/lib:$ORAWEB_HOME/../cartx/common/perl/lib/site_perl/sun4-
solaris:$ORAWEB_HOME/../cartx/common/perl/lib/site_Perl
```

Perl の拡張モジュールの移行

すでに Perl インタプリタを持っていて、その環境用の拡張モジュールを開発している場合があります。同じ拡張モジュールを Perl カートリッジ環境で使用できるようにするには、モジュールをカートリッジ環境で作り直す必要があります。このために、Perl カートリッジに入っている Perl インタプリタを使用することができます。

Perl の拡張モジュールを Perl インタプリタのインストール環境から Perl カートリッジのインストール環境に移動するには、次のステップを実行します。

1. 次の状況を想定します。
   - Perl インタプリタがインストールされているディレクトリを PI とします。
   - Perl カートリッジがインストールされているディレクトリを PC とします。このディレクトリは **$ORAWEB_HOME/../cartx/common/perl** です。
   - "Mod" が Perl モジュール名で、モジュール名と同じ名前のディレクトリが **PI/perl/lib/site_perl/sun4-solaris/auto/** にあるとします。すなわち、ディレクトリ **PI/perl/lib/site_perl/sun4-solaris/auto/Mod** があるとします。
   - **PI/lib/site_perl/Mod.pm** ファイルがあるとします。
2. **PI/perl/lib/site_perl/sun4-solaris/auto/Mod** ディレクトリを **PC/lib/site_perl/sun4-solaris/auto/** にコピーします。
3. **PI/lib/site_perl/Mod.pm** ファイルを **PC/lib/site_perl** ディレクトリにコピーします。
4. ステップ 2 で移動したすべてのモジュールを反映するために、**PC/lib/sun4-solaris/5.00401/perllocal.pod** ファイルを更新します。**perllocal.pod** ファイルには、ご使用の環境にある各モジュールの情報が含まれます。**PI/lib/sun4-solaris/5.00401/perllocal.pod** ファイルをファイルの更新方法の参考としてご使用ください。

Perl での Oracle Application Server API へのアクセス

Oracle Application Server API にアクセスする Perl の拡張モジュールを開発できます。一般的には、モジュール内に Oracle Application Server API のラッパーを作成し、Perl で Web アプリケーションを開発する場合の Perl 言語の一部と同じインタフェースを使用できるようにします。ほとんどの API コールでは、1 番目の引数に WRBContext を使用する必要があります。Perl カートリッジを使用すると、Perl のグローバル変数 "::WRB" を介してカートリッジの WRBContext にアクセスできます。モジュールを C で開発する場合は、次のようにして WRBContext にアクセスできます。

```
void *WRBCtx;
WRBCtx = SvIV(perl_get_sv("::WRB", FALSE));
```

14

Perl インタプリタのアップグレード

Perl カートリッジと LiveHTML カートリッジに付属している Perl は、パブリック・ドメインから入手可能な Perl と同じです。カートリッジとともにインストールされる Perl のバージョンは、5.004_01 です。新しいバージョンにアップグレードする場合は、この章の手順に従ってください。Perl の最新バージョンは **http://www.perl.com** からダウンロード可能です。

注意： 5.004_01 より古いバージョンの Perl インタプリタは、Oracle ではサポートされていません。

内容

新しいインタプリタのインストール

Perl インタプリタは、共有ライブラリとしてインストールする必要があります（UNIX の場合、Perl インタプリタは `libperl.so` としてインストールされます）。

Perl インタプリタの最新バージョンをダウンロードした後、そのファイルを一時ディレクトリに解凍します。設定ユーティリティを、次のように `-Duseshrplib` オプションを使用して実行します。

```
prompt> Configure -Dprefix=<installation directory> -Duseshrplib
```

指示に従います。一時ディレクトリで make コマンドを実行して、インストール・プロセスを続行します。UNIX の場合、次のように入力します。

```
prompt> make
prompt> make test
prompt> make install
```

NT の場合、次のように入力します。

```
prompt> nmake
prompt> nmake test
prompt> nmake install
```

ダウンロード・ファイルに付属の README ドキュメントに、詳しいインストレーション情報が入っています。

<installation directory> については、Perl または LiveHTML アプリケーションの環境変数にパスが反映されていれば、Oracle Application Server 以外のパスを指定できます（次の項を参照）。Oracle Application Server と同時にインストールされるインタプリタは、UNIX の場合 $ORACLE_HOME/ows/cartx/common/perl、NT の場合 %ORACLE_HOME%¥ows¥cartx¥common¥perl¥ に入っています。

新しいインタプリタを使用するためのアプリケーションの設定

インタプリタのインストールが完了したら、新しいインタプリタを使用する Perl および LiveHTML アプリケーションの設定情報を変更する必要があります。具体的には、LD_LIBRARY_PATH（NT の場合 PATH）および STD_PERLLIB の 2 つの環境変数を変更する必要があります。

LD_LIBRARY_PATH（UNIX）または PATH（NT）の値を変更するには、次のステップを実行します。

1. Oracle Application Server Manager で、Perl または LiveHTML アプリケーションの「設定」フォルダの「環境変数」フォームを開きます。
2. UNIX の場合、LD_LIBRARY_PATH 設定を探し、

   ```
   %ORACLE_HOME%/ows/cartx/common/perl/lib/sun4-solaris/5.00401/CORE
   ```

   という記述を次の記述に置き換えます。

   <new installation directory>/perl/lib/sun4-solaris/*<new version>*/CORE

   NT の場合、PATH 設定を探し、

   ```
   %ORACLE_HOME%¥ows¥cartx¥common¥perl¥bin
   ```

   という記述を次の記述に置き換えます。

   <new installation directory>¥bin

STD_PERLLIB の値を変更するには、次のステップを実行します（UNIX、NT とも共通）。

1. シェルまたは DOS のプロンプトで、それぞれ次のコマンドを実行します。

   UNIX

   ```
   prompt> perl -e 'print join(":", @INC)'
   ```

   NT

   ```
   prompt> perl -e 'print join(";", @INC)'
   ```

2. Oracle Application Server Manager のウィンドウで、「リロード」ボタンをクリックします。

   「環境変数」フォームの STD_PERLLIB 設定に、新しいインタプリタの設定が反映されます。

15

トラブルシューティング

Perl スクリプトの実行に関する問題

Perl スクリプトが実行できない場合には、次の事項を確認してください。

- カートリッジの仮想パスが Perl スクリプトが入っている物理ディレクトリにマップされているかどうか確認します。
- Oracle Application Server が正常に機能しているかどうか確認します。これをチェックするには、他の Perl スクリプトやカートリッジを実行してみてください。サンプルの Perl スクリプトを実行することもお薦めします。

ログ・ファイル

Perl カートリッジのロギングが使用可能に設定されている場合、「ロギング」ページ（「設定」の下にある）で指定したファイルにログ・メッセージが書き込まれます。

さらに、Perl カートリッジは、Perl スクリプト内から stderr に送ったメッセージ（たとえば、`warn` や `die` からの出力）もログ・ファイルに書き込みます。

未処理のエラー

Perl カートリッジは、`eval` が実行されるサブルーチンとして Perl スクリプトを実行します。スクリプトでエラーが発生すると、`eval` から Perl カートリッジにエラーが返されます。これにより、エラーがログ・ファイルに書き込まれ、ブラウザにエラー・メッセージが送信されます。

索引

記号

A

C

E

F

H

I

L

M

N

O

P

Q

R

S

T

し

す

せ

そ

ち

て

と

な

ね

は

ふ

み

も

ら

ろ