株式会社　宇部情報システム御中

宇部興産殿向け
新石炭総合OAシステム
サーバー・アプリケーション
アーキテクチャ設計書

Ver 1.0

平成１１年１月２９日（金）

株式会社　オージス総研

1 目的

本書は「宇部興産殿向け 新石炭総合OAシステム」において構築したサーバーアプリケーションのアーキテクチャ（＝基本構造）について記述する。

2 目標

今回のアプリケーション構造を設計するに当たっては、次の3点を目標とした。

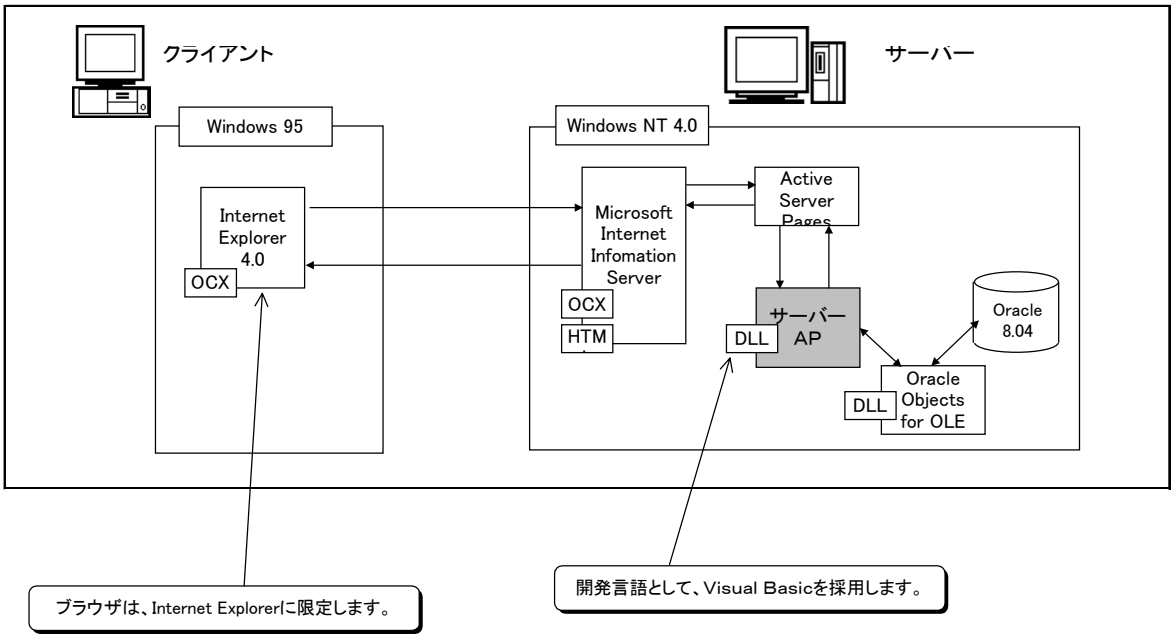
① 見通しの良いソフトウェア構造の実現
全体として見通しが良く、各クラスの役割分担が明確となるようなアプリケーション構造を実現する。
このことにより、アプリケーションの保守性、拡張性、理解容易性の向上を図る。

② 信頼性の確保
クライアントAPにとって使いやすく、かつ信頼性の高いサーバーAPを構築する。

③ 特定製品に固有なインタフェースの隠蔽
特にクライアントAPに対して、特定製品に固有なインタフェースに依存する部分を少なくすることで、製品のバージョンアップや入れ替えを行う際の影響範囲を局所化する。

3 システム構成

今回のシステム構成を以下に記述する。



■クライアント

クライアントPCへのソフトウェア配布による運用負荷を軽減するため、基本的にはブラウザの搭載だけを前提とする。ただし一部のユーザーインタフェースで、Windows ベースの使い勝手が要求されているため、Internet Explorer 4.0 を前提として、ActiveX コンポーネントを利用することが想定されている。

■サーバー

サーバー上に搭載する主要なソフトウェアは次の通り。

- OS :Windows NT 4.0
- Web サーバー:Microsoft Internet Information Server (IIS)
- DBMS :Oracle 8.0.4

（MTS (Microsoft Transaction Server)については、今回は十分な調査期間が取れなかったことから採用を見送った。）

■開発言語
サーバー・アプリケーションの開発言語は Visual Basic 6.0 を採用する。

以降では、Rational Unified Process でのアーキテクチャの定義（4＋1ビュー）に従って、5つの視点からアーキテクチャを記述する。

- ユースケース・ビュー
- 論理ビュー
- 並行性ビュー
- 配置ビュー
- 実装ビュー

4 ユースケース・ビュー

本書はアプリケーションの基本構造を記述することを目的としているため、ユースケース全体像についてはここでは記述しない。（関連ドキュメントを参照のこと。）
アーキテクチャ構築を行う上で対象としたユースケースは次の通りである。

- 銘柄メンテナンス（登録、変更、削除、検索）
- 基本契約メンテナンス（登録、変更、削除、検索）

これらのユースケースを選択した理由は次の通り。

① 機能の網羅性
今回のシステムでは永続オブジェクトの種類として、1)マスター 2)トランザクションに加えて、3)メモリ上に情報を保持する形態の3種類がある。上記のユースケースを実装することでこの3種類の永続オブジェクトの形態を網羅できる。
また上記の分類とは別に、永続オブジェクトの中には、ヘッダー／明細形式のものがあるが、基本契約がこれに相当する。
これらのことから選択したユースケースをまず実装することにより、残りのユースケースを開発する際に安定したアーキテクチャ（アプリケーション基盤）を構築できると判断した。

②重要性
業務的にもっとも優先度の高いのは売上サブシステムと基本共通サブシステムであるが、上記のユースケースはこの一部に含まれている。

③効率
売上サブシステムは、クライアント数ももっとも多く、パフォーマンスのボトルネックとなることが想定されている。したがってこの部分を先に作ることで、システム全体として効率問題がないことを早期に検証すること意図されているが、この目的にも合致する。

5 論理ビュー

ここでは、アーキテクチャを構成するパッケージとクラスの説明を記述する。
本来「論理ビュー」では、実装環境に依存しない論理的なクラス構造を指すものである。
しかし説明の都合上から、Visual Basic の実装構造に依存したクラス（および標準モジュール）についても併せてここで記述する。

5.1 基本方針

①信頼性の確保
オペレータから入力された内容の妥当性チェックは基本的にすべてサーバーAPで行う。
これにより、クライアントAPの構造を極力単純にすると同時に、データベースに不整合なデータを書き込むことを防止する。

②エラー処理

エラーは、次のように分類して考える。

- 致命的エラー（ソフトウェアバグ）
- 運用エラー（オペレータの入力ミス）

致命的エラーが発生した場合、エラーの原因追及ができるように、エラーの発生箇所とエラー情報を Err オブジェクトに格納して、Visual Basic の例外を発生させ、クライアントに後処理を任せる。

運用エラーの場合も同様に、Visual Basic の例外を発生させ、クライアントに後処理を任せる。

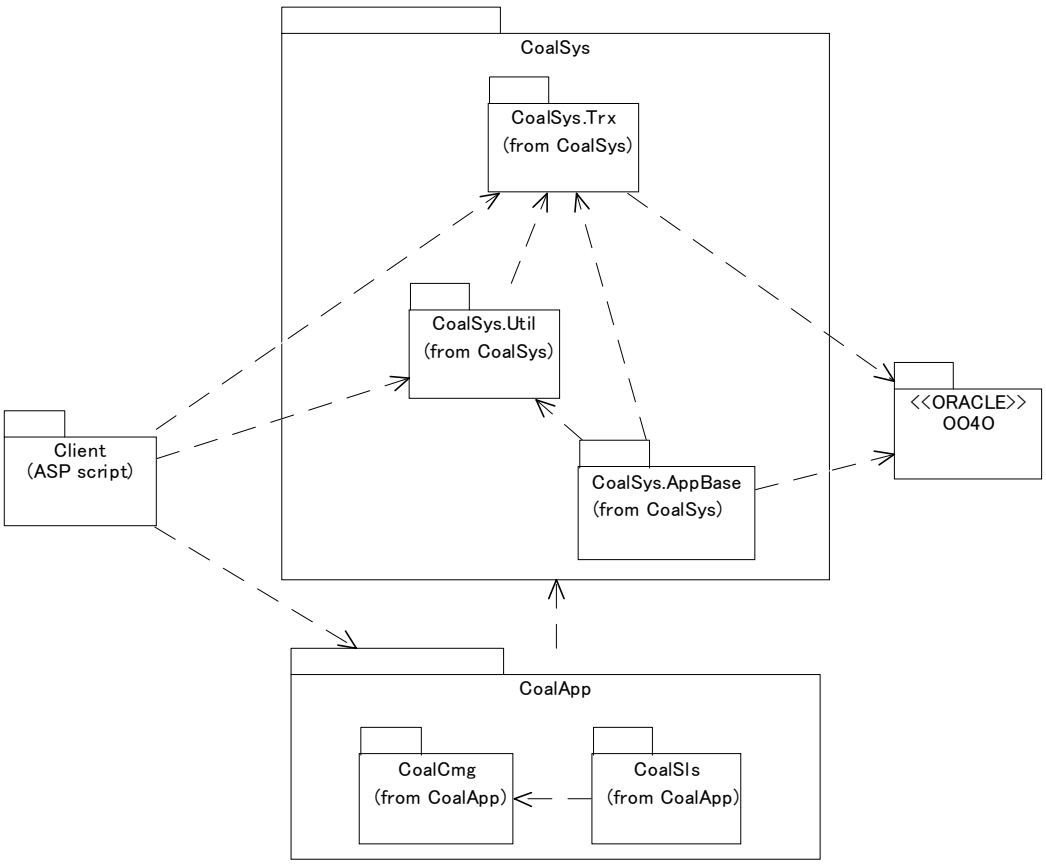
本来ならばクライアントAP側でエラーの種類の判定を容易にするために、発生させる Err オブジェクトの種類を変えたいところだが、Visual Basic では、Err クラスを（継承により）拡張できないため、エラー番号でエラーの種類を識別するものとする。

③特定製品に固有なインタフェースの隠蔽

Oracle データベース・コンポーネントのような特定の製品固有のインタフェースをクライアントAPから隠蔽する。

これにより製品を入れ替えた場合に、アプリケーションに対する影響度を少なくする。

パッケージ構成

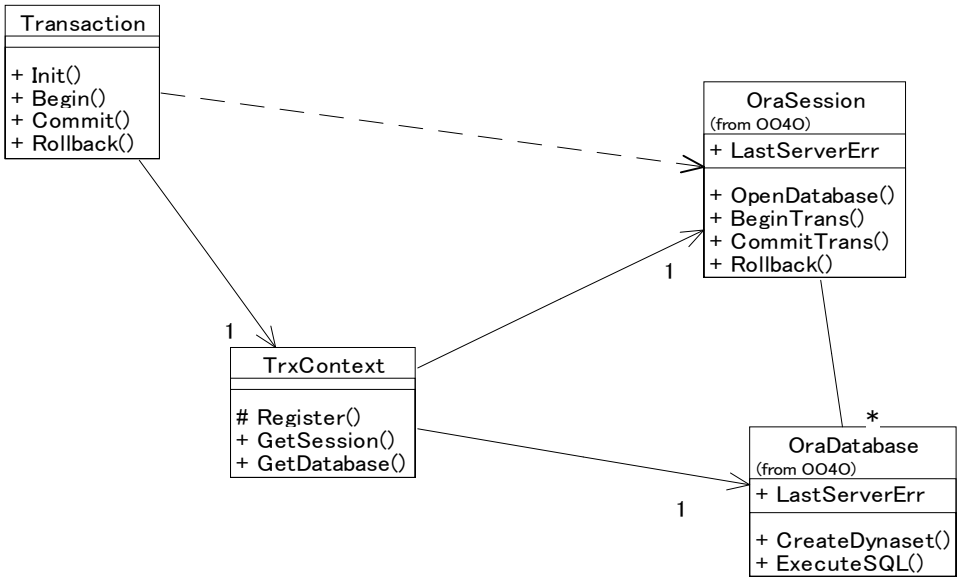


<< パッケージの説明 >>

名称	パッケージ名	説明
システム共通	CoalSys	共通機能、ユーティリティの総称
トランザクション	CoalSys.Trx	トランザクション制御
ユーティリティ	CoalSys.Util	各種ユーティリティ
AP基本	CoalSys.AppBase	アプリケーションの基本となるクラス群
Oracle Objects for OLE	OO4O	Oracle で提供されるコンポーネント
アプリケーション	CoalApp	サーバーAPの総称
クライアント	Client	クライアントAPの総称（Visual Basic による開発対象外）

5.2 トランザクション（CoalSys.Trx）パッケージ

トランザクション制御をするクラスを集めたパッケージ。実際のトランザクション制御は OO4O (Oracle Objects for OLE)で提供される Session オブジェクトを使用して制御するが、クライアントAPには特定製品固有のインタフェースを隠蔽するために、次のクラスを定義した。



5.2.1 Transaction クラス

OO4O の Session オブジェクトのラッパー・クラス。
トランザクションを開始(Begin)、終了(Commit)、異常終了(RollBack)するためのメソッドを提供している。
クライアントAPは、このオブジェクトを通じてトランザクション制御を行う。

またデータベース情報を初期設定(Init)するためのメソッドも提供している。ここでは、データベースの位置、およびログインユーザー名とパスワードをクライアントAPから指定させており、Oracle に依存する仕様となっている。
しかしこれらの情報をサーバーAP内でローカルに保持し、この Init メソッドを公開しない仕様にした場合、データベース位置やユーザー名、パスワードを変更する度にサーバーAPを再コンパイルする必要が出てしまうため、このような仕様とした。

5.2.2 TrxContext クラス

トランザクションのコンテキスト情報を保持するクラス。
具体的に保持する情報は次の通りである。

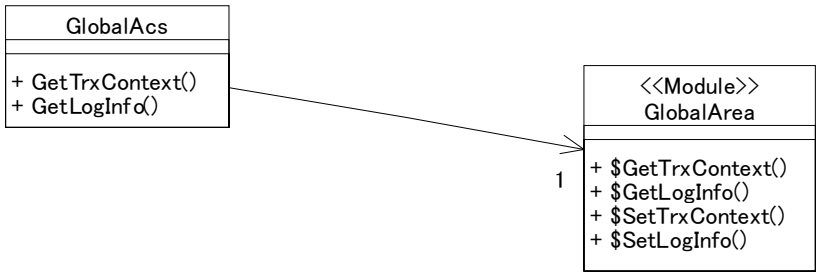
① OraSession オブジェクト
② OraDatabase オブジェクト

この TrxContext オブジェクトは、スレッド内グローバル領域(GlobalArea)に保持しておく。
これにより、コンテキスト情報が必要なオブジェクト(永続オブジェクトなど)は、必要なときにいつでも取り出すことが可能になる。

5.3 ユーティリティ（CoalSys.Util）パッケージ

5.3.1 スレッド内共通領域

トランザクションコンテキスト情報、ログ情報のように、種々のオブジェクトから共通に参照される情報を格納するためのスレッド内グローバル領域、および、それをアクセスするためのクラス群から構成される。



5.3.1.1 グローバル領域 (GlobalArea)

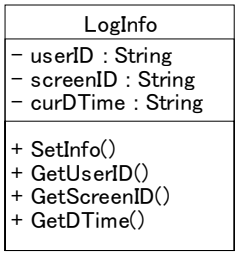
スレッド内グローバル領域を定義した標準モジュール。
（C++や JAVA などの一般的なオブジェクト指向言語の場合、static 変数を利用して、Singleton オブジェクトを作るようにすればこのような仕組みは必要ないが、Visual Basic では static 変数ができないため、代わりに標準モジュールを利用する必要がある。）
具体的にグローバル領域で保持されるオブジェクトは、次の2つである。

① トランザクション・コンテキスト情報(TrxContext)
② ログ情報(LogInfo)

5.3.1.2 グローバル領域アクセス用クラス (GlobalAcs)

上に書いたように、グローバル領域のために標準モジュールを使用したが、さらに異なるプロジェクトからは標準モジュールにはアクセスできない、という制約がある。
今回のシステムでは、ソフトウェアの成果物管理を容易にすることなどから、アプリケーション全体を1つのプロジェクトにはせず、複数のプロジェクトに分割している。
したがって、グローバル領域(GlobalArea)を含まないプロジェクトからもアクセスを可能とするために、専用のクラスとしてグローバル領域アクセス用クラス(GlobalAcs)を用意した。

5.3.2 ログ情報 (LogInfo)

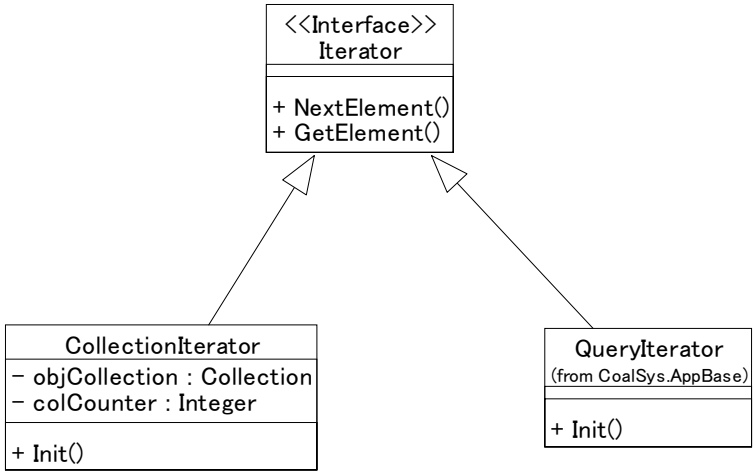


DBに格納するログ情報は、次のものから成る。

① ログインユーザー
② 画面名称
③ 日付、時刻情報
④ 処理区分（新規登録、修正、削除）

上記のうち、③④はサーバーAP内のオブジェクトが自動的にセットすることができるが、①②はサーバーAPでは取得することができず、クライアントAPからサーバーに教える必要がある。これを各ドメインオブジェクトごとに毎回教えるインタフェースでは煩わしくなるため、1トランザクションごとにクライアントAPからサーバーAPに1回だけ教えるようにした。このために LogInfo オブジェクトを用意した。

5.3.3 イテレーション（繰り返し処理）関係



5.3.3.1 イテレーション基本インタフェース (Iterator)

オブジェクトの条件指定による検索（クエリー）などの場合で、クライアント側に複数のオブジェクトを一括して返す場合の共通インタフェースとして、Iterator を定義した。
クライアントAPは、このインタフェースを利用することで、イテレーションの内部構造を意識する必要がない。

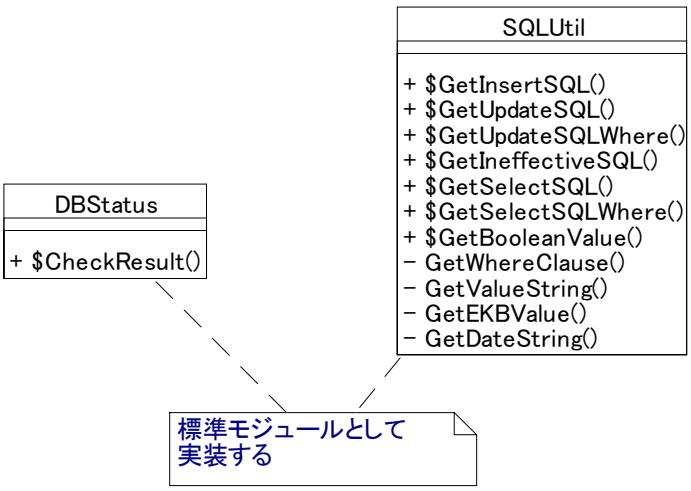
5.3.3.2 クエリー用イテレータ (QueryIterator)

条件指定などによる RDB 検索の読み込み結果からイテレーション処理をするためのオブジェクト。
検索では大量レコードが処理対象になる可能性があり、それらのオブジェクトをすべて一括して生成しようとすると、メモリリソースの大量消費や、レスポンスタイムの悪化といった深刻な効率問題につながる危険がある。
したがって DB の読み込み結果をイテレーションするときに、クライアントから読み込み要求があったタイミングでオブジェクトを生成するようにし、かつそれ以前読み込んだ古いオブジェクトは、メモリから解放されるようにするため、この QueryIterator を用意した。

5.3.3.3 コレクション用イテレータ (CollectionIterator)

VBA (Visual Basic for Applications)標準のビルトイン・クラスである Collection オブジェクトのラッパー。
ASP スクリプトでは、Collection オブジェクトを利用した For Each 命令を記述することができないこと、およびクライアントAPにイテレーションの内部構造を意識させないことの、2つの理由から Iterator インタフェースを継承したこの CollectionIterator を用意した。

5.3.4 SQL 関係



5.3.4.1 SQL ユーティリティ (SQLUtil)

SQL の文字列を生成するロジックは単純だが、カンマやスペースを配置する規則が微妙であり、間違えを引き起こしやすい。また、もしエラーがあってもコンパイルでは検出できない。したがってこのロジック記述を容易にし、かつ信頼性を高めるために次のような関数を提供するクラスユーティリティとして SQLUtil を用意した。

① INSERT 文を生成する関数
② UPDATE 文を生成する関数
③ SELECT 文を生成する関数
④ 無効化する SQL 文を生成する関数

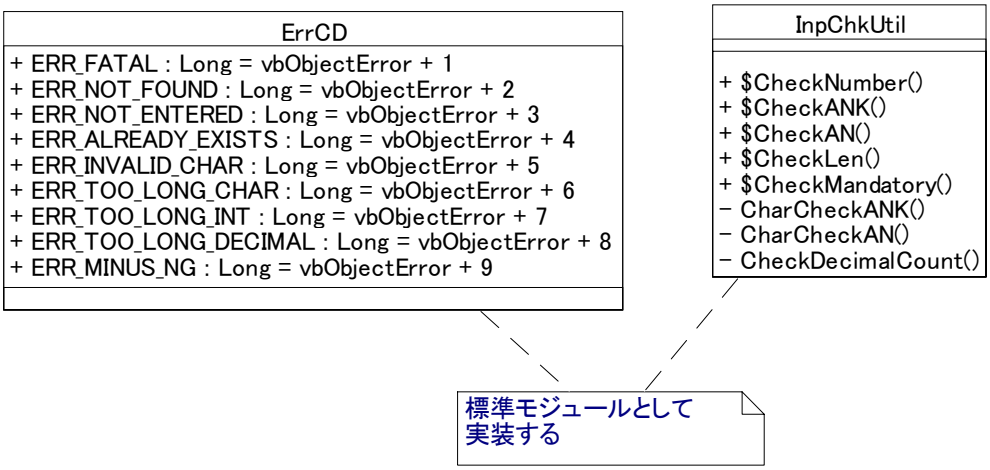
さらに、データベースに格納する際にシステム全体で共通となる次のような機能のサポートを加えた。

① 登録・更新時のログ情報用の SQL 生成
② Boolean 属性の文字列変換
③ String 文字列の調整（後続スペースのカット、および全スペースの時の１文字スペース変換）

5.3.4.2 DB ステータス判定 (DBStatus)

Oracle のエラーコードを解析して、致命的エラーか、運用上起こりうるエラーかを判定するためのクラスユーティリティ。Oracle 固有のエラーコードを判定する部分を局所化するため、このクラスユーティリティを導出した。
具体的には、INSERT 文を実行した時に、同一キーのレコードがすでに存在していた場合、キー情報が手採番であれば、致命的エラーではなく、オペレータの入力ミスとする必要がある。
しかしこのエラーコードは Oracle 固有のものであるため、このクラスユーティリティで提供されるメソッドで判定する。

5.3.5 業務ユーティリティ



5.3.5.1 入力チェック用クラスユーティリティ (InpChkUtil)

今回のアプリケーションでは、入力された項目値の妥当性のチェックはサーバー側のドメイン・オブジェクトがすべて受け持つ。（パフォーマンスや操作性などの事情がある場合は、クライアント側でも必要に応じてチェックをすればよい。）
これらの入力内容の妥当性をチェックするメソッドを集めたクラスユーティリティとして InpChkUtil を用意した。

① 数値項目と桁数チェック (CheckNumber メソッド)
② 英数字カナ項目チェック (CheckANK)
③ 英数字項目チェック (CheckAN)
④ 文字列の長さチェック (CheckLen)
⑤ 必須入力チェック (CheckMandatory)

5.3.5.2 エラーコード定義用クラスユーティリティ (ErrCD)

5.1で記述した、エラーの種類を判定するためのエラー番号を定義したクラスユーティリティ。

5.4 AP基本（CoalSys.AppBase）パッケージ

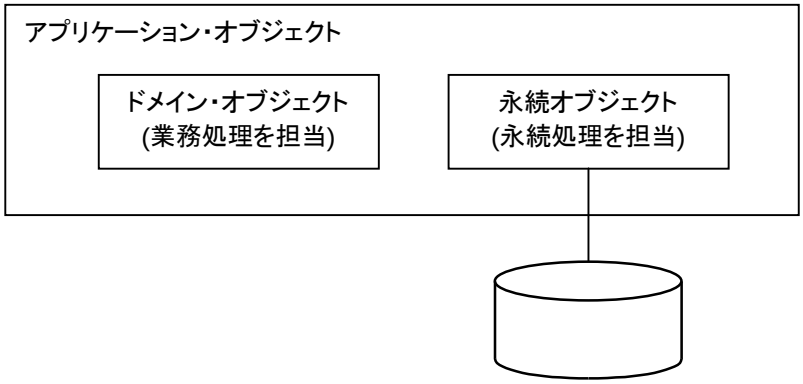
5.4.1 ドメイン・オブジェクトと永続記憶のマッピング

基本的には、1つのドメイン・クラスに対して、1つのRDBのテーブルを定義し、属性ごとにカラムを割り当てる。

ただし、業務運用上で内容が変更されることがないマスター情報については、オブジェクトのプロパティ情報を集めた Collection オブジェクトとしてメモリ上に保持する。

5.4.2 業務処理と永続処理の分離

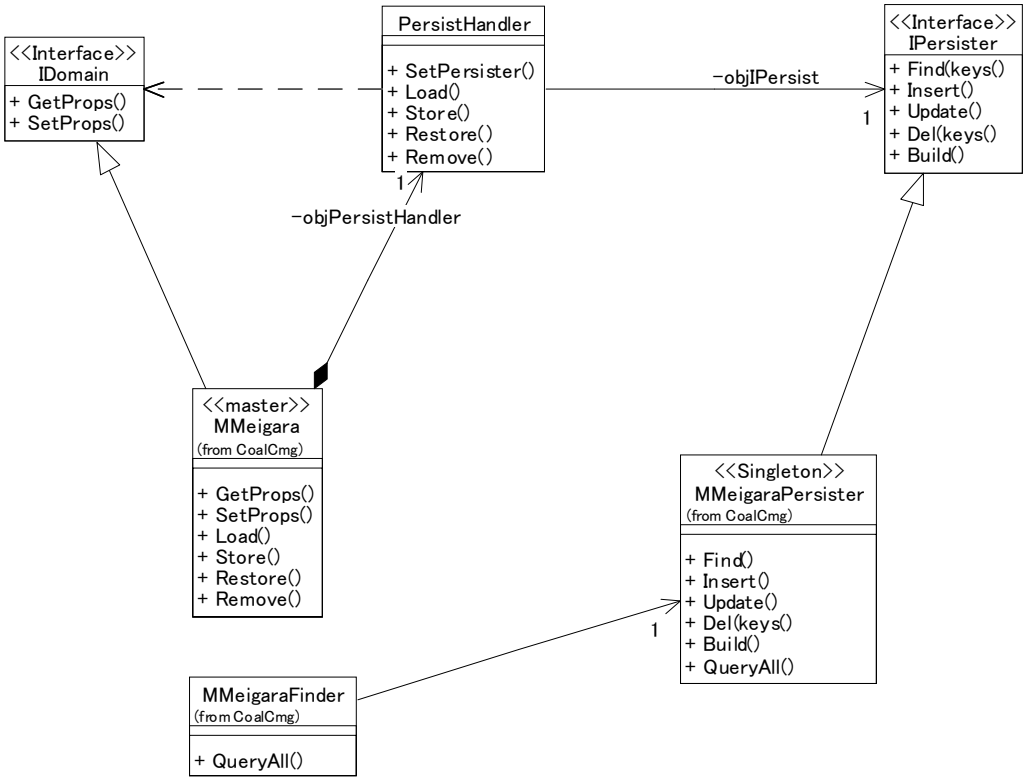
アプリケーション・オブジェクトは大きく、業務処理を担当するドメイン・オブジェクト(Domain)と、永続記憶へのアクセス処理を担当する永続オブジェクト(Persister)の2つに分割する。



これにより、次のような効果が得られ、結果としてアプリケーションの理解が容易となり、保守性、拡張性を向上させることを意図した。

- オブジェクト指向モデルと親和性があまり高くないリレーショナルモデルとの意味的ギャップを Persister オブジェクト側に隠蔽する。
- 障害解析や監査証跡などの目的で採取するデータベースの履歴情報や赤黒伝票処理などを Persister オブジェクト側に任せることで、Domain オブジェクトを本来の業務仕様の処理だけに専念させる。
- RDB、メモリやファイルなど、オブジェクトを格納する記憶媒体が変更されても、業務処理を担当する Domain オブジェクトに影響を及ぼさない。

また、業務処理と永続処理を分離する構造をサポートする簡単なフレームワークを定義して使用を強制することで、アプリケーション全体構造が一貫するように配慮した。
基本となるクラス構造は次の通りである。
（注：インタフェースとそれを実装するクラスの関係は、本来は洗練(refinement)関係であるが、ここでは継承(inheritance)関係として記述している。）



以下に各ラスの説明を記述する。

① ドメイン共通インタフェース (IDomain) [1]
業務処理に責任を持つドメイン・クラスが実装しなければならないインタフェース。
プロパティ情報を取得、および設定するためのメソッドが定義されており、これらのメソッドは PersistHandler オブジェクトから呼び出される。

② 永続処理ハンドラ (PersistHandler)[1]
ドメイン・オブジェクトと永続オブジェクトの相互作用をするメソッドが定義されている。

③ 永続処理共通インタフェース (IPersister)
DBなどの永続記憶へのオブジェクトの格納、読み出しに責任を持つ永続クラスが実装しなければならないインタフェース。
オブジェクトを永続記憶に新規登録、更新、削除する、あるいは永続記憶から検索したり、オブジェクトのプロパティ情報を復元するためのメソッドが定義されている。

以降は、個々のドメインオブジェクトの種類ごとに作成されるクラスである。

④ ドメイン・クラス (MMeigara 他)
業務処理に責任を持つクラス。
クライアントAPから渡されたデータ内容のチェックは、このクラスの責任である。

⑤ クエリー受付クラス (MmeigaraFinder 他)
条件指定によるクエリー処理の窓口になるクラス。
基本的にはクエリーの種類ごとにメソッドを用意する。

---

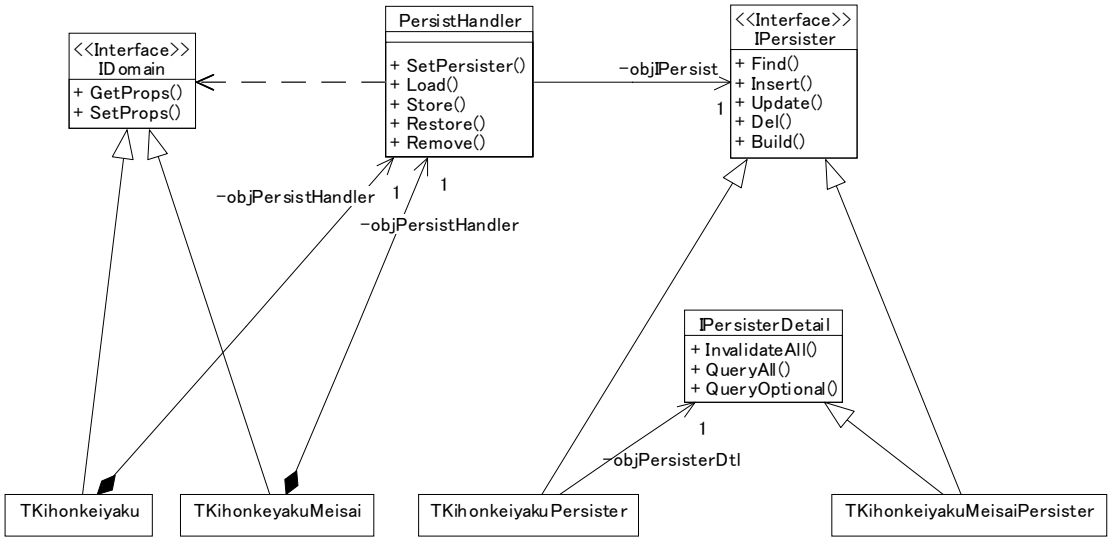
[1] 本来ならば、ドメイン・オブジェクトが永続オブジェクトとやり取りをする部分をスーパークラスとして定義し、各ドメインオブジェクトでサブクラス化する仕様としたかったが、Visual Basic では実装の継承機能がサポートされていないため、インタフェースとしての IDomain と継承すべきメソッドを委譲で実現する PersistHandler の2つのクラスを導出している。

このクラスは、永続処理クラスをクライアントAPから隠蔽するために用意したもので、自身では基本的に何もせず、永続処理クラスに処理を委譲する。

⑥ 永続処理クラス (MMeigaraPersister 他)
Persister オブジェクトは、状態を持たないため、メモリ節約のため、スレッド内 Singleton として実装する。
履歴レコードの管理、および赤黒伝票処理、ログ情報の付与などは、基本的にこのクラスの責任である。

5.4.3 ヘッダー／明細オブジェクト

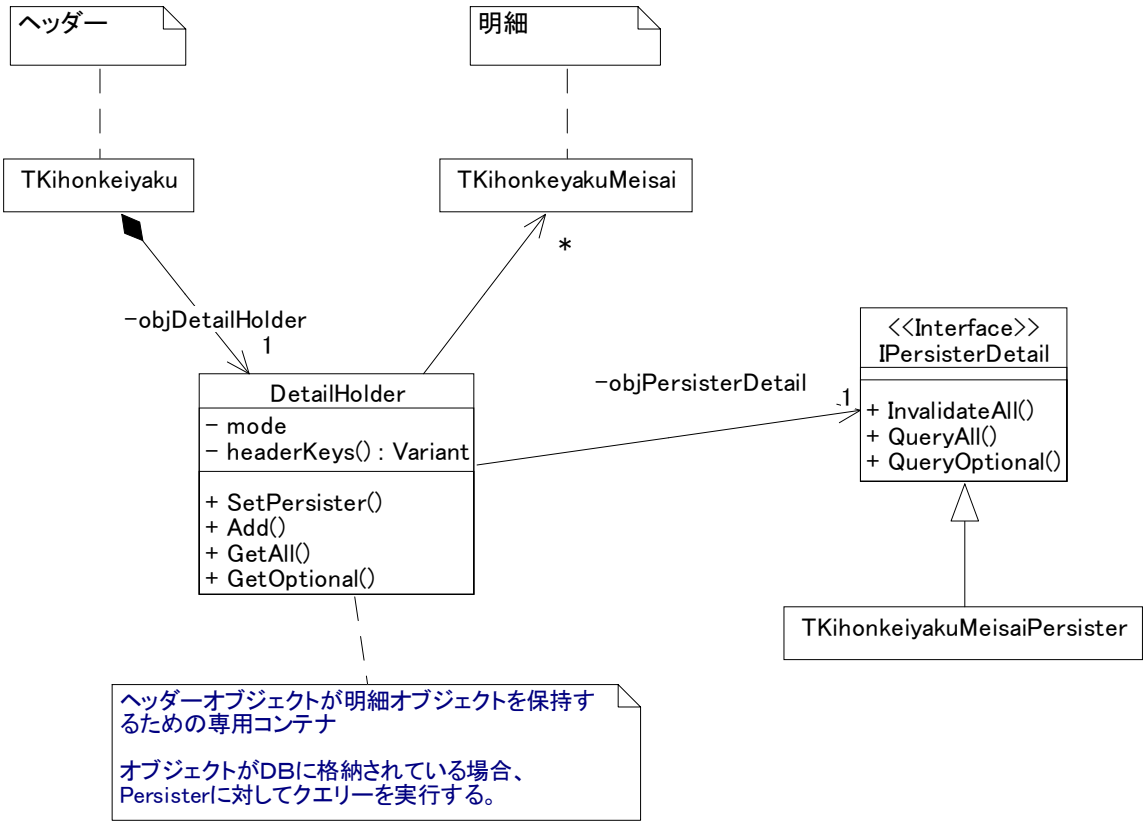
基本契約、売上などのヘッダー／明細構造を持つオブジェクトの基本的なクラス構造を以下に示す。



基本的には、5.4.2で記述した内容と同じだが、明細一括処理用の永続インタフェースを追加した。

⑦ 明細一括処理用の永続インタフェース (IPersisterDetail)
ヘッダー・明細形式では、更新や削除要求があった場合は、明細レコードは一括して無効化をする必要がある。（パフォーマンスの考慮に加えて、更新時には明細レコードの件数が増える可能性があるため、古い明細の無効化処理は明細オブジェクトの責任にはできないため。）
したがって、明細レコードを一括して無効化やクエリーするためのインタフェースを用意した。

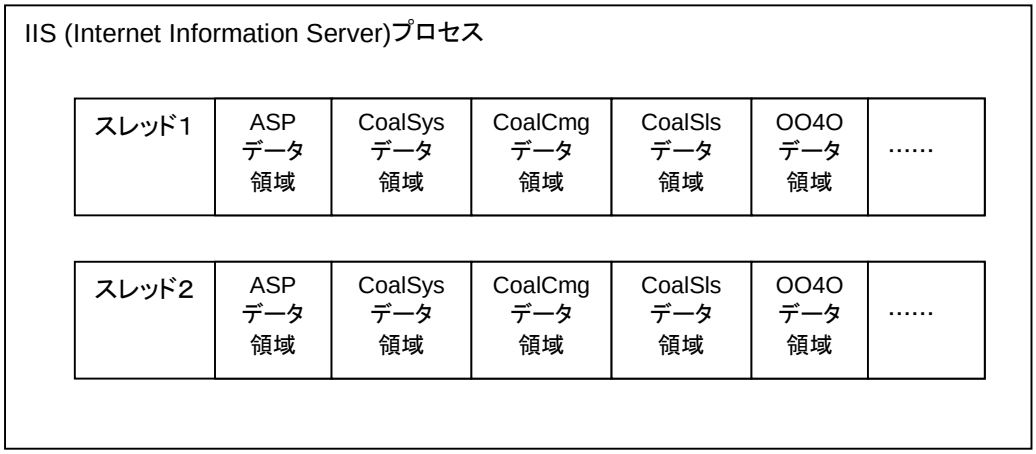
また、上記に加えて、ヘッダーオブジェクトが明細オブジェクトを保持するためのコンテナオブジェクトとして、DetailHolder を用意した。



⑧ 明細ホルダー(DetailHolder)
ヘッダーオブジェクトが明細オブジェクトを保持するための専用コンテナ。
ヘッダーオブジェクトが永続記憶から読み込まれていた場合（Load された場合）は、このDetailHolderにキー情報と、明細Persisterオブジェクトを教えることで、DetailHolderが自動的にDBに対するクエリーを実行する。

6 並行性ビュー

今回開発したサーバーアプリケーションは、マルチスレッド対応のダイナミックリンクライブラリ(DLL)であり、スレッド構成はクライアントアプリケーションである IIS (Internet Information Server)の構造に依存する。
しかしVisual Basicで作成できるサーバーAPは、基本的にアパートメント・スレッドであり、スレッド間で共有されるデータ領域はない。
またスレッド間通信も基本的に発生しないため、次のような構造になると考えられる。



7 配置ビュー

システム構成を参照のこと。

8 実装ビュー

8.1 実装構造(プロジェクトの設定)

サーバーアプリケーションは ActiveX DLL として作成する。
Visual Basic のプロジェクト・プロパティに設定するオプションは次の通り。

- スレッドモデル　　アパートメントスレッド

8.2 プロジェクトの単位

Visual Basic の開発環境に定義するプロジェクトは、開発運用管理の利便性と、プロジェクトの単位でコンポーネント(DLL ライブラリ)になること考慮して、次のようにした。

- システム共通 (CoalSys) 関係については、全体を一つにまとめる。
- アプリケーションについては、売上、運賃、受払などのサブシステム毎にプロジェクトを作成する。

現時点(99/01/29)で作成したプロジェクトは次の通り。

① システム共通　　CoalSys
② 業務共通　　CoalCmg
③ 売上　　CoalSls

8.3 ディレクトリ構成

ディレクトリ構成は上記と同じだが、次の理由により、さらに各プロジェクトで共有するモジュールを格納する Global ディレクトリを作成した。

■Global ディレクトリが必要な理由
Visual Basic ではプロジェクト間にまたがるやり取りは、クラスモジュールを介してしかできない。
Visual Basic のクラスモジュールでは、定数を含むクラス変数やクラスメソッドを定義できないという制約があるため、状態を持たない単なる関数群やコンスタント集は、標準モジュールとして作成し、Global ディレクトリに格納して、各プロジェクトに含めることとした。

8.4 論理ビューとの対応関係

基本的には、クラス＝クラスモジュールとして実装する。しかし状態を持たない関数だけを定義したクラスユーティリティは、標準モジュールとして実装する。
Persister クラスは、Singleton オブジェクトとして実装する。
Singleton を実装するためには、クラススコープの関数を用意する必要があるが、これは標準モジュールとして作成する必要があるため、別に用意した。また Domain と Persister では、属性情報を共有する必要があるが、そのためのユーザー定義型を格納する情報も併せて xxMod.bas という標準モジュールを作成した。
したがって、xxx という名前のドメインクラスがある場合、次の4つのモジュールが作成されることになる。
xxx.cls
xxxPersister.cls
xxxMod.bas
xxxFinder.cls

8.5 クラスモジュールの Instancing プロパティの設定

Visual Basic では外部プロジェクトからのクラスの可視性を Instancing プロパティで設定する。ActiveX DLL としてプロジェクトを作成した場合、クラスモジュールに対して 4 種類の Instancing プロパティを設定することができる。以下に Instancing プロパティの説明を示す。

<< Instancing プロパティの説明 >>

Instancing プロパティ	説明
1 – Private	クラスが属するプロジェクトの内部でのみ、インスタンスを生成可能でかつ、使用可能なクラス
2 – Public Not Creatable	外部のプロジェクトからのインスタンスの生成が不可能であるが、使用可能なクラス
5 – MutiUse	外部のプロジェクトからもインスタンスの生成可能でかつ、使用可能なクラス
6 – GlobalMultiUse	ユーザーがクラスの生成を行う必要がなく、いつでも使用可能なクラス

今回作成したクラスモジュールに関しては、以下の方針で Instancing プロパティを設定した。

- プロジェクト内部でのみインスタンスが使用されるクラスは、1 - Private を設定する。
  永続化処理クラスなど。
- インタフェースのような生成する必要のないクラスは、2 - Private Not Creatable を設定する。
  IDomain や IPersistar など

- その他のクラスに対しては 5- MultiUse を設定する。
  ドメイン・クラスや LogInfo など

8.6 Singleton の実装

Visual Basic では、C++や JAVA でいう static メソッドを定義できない。そこで今回の Singleton では、オブジェクトを格納するプライベート変数を標準モジュールで定義し、そのプライベート変数にアクセスするための手段として標準モジュールの関数を提供するという実装とした。

9 データモデル

基本的にはドメインオブジェクトをそのままRDBに格納した。

9.1 オブジェクトの属性と RDB のマッピング規則

基本的にオブジェクトの属性の型を変換せずに RDB に格納できるように RDB のデータ型を決定した。オブジェクトの属性とRDBのデータ型のマッピング規則を以下に示す。

<< オブジェクトの属性の型と RDB のデータ型のマッピング規則 >>

オブジェクトの型	RDB のデータ型
Integer / Long / Double	NUMBER
String	VARCHAR2
String(日時を表す場合)	DATE
Boolean	VARCHAR2

■属性が数値型(Integer/Long/Double)
Visual Basic の数値型を RDB に格納する場合には、RDB のデータ型は NUMBER 型とする。
■属性が String 型
Visual Basic の String 型では、文字は内部で Unicode として扱っている。そのため String の長さを指定した場合には、格納される文字が 1byte 文字か 2byte 文字かということにかかわらず、指定した文字分文字を格納することができる。また、RDB の VARCHAR2 型では文字数ではなく byte 数を指定する。そこで、String 型の属性は漢字(2byte 文字)が入力される可能性がある場合、VARCHAR2 型の半分の長さとする。
■属性が String 型(日時を表す場合)
今回のシステムでは、DATE 型による検索が必要である。そこで、日時を RDB に格納する際、String 型から DATE 型への変換を行っている。この変換は SQL の TODATE を使用する。
■属性が Boolean 型
Visual Basic の Boolean 型を RDB に格納する場合、RDB のデータ型には対応する型が存在しない。そこで Boolean 型を String 型に変換し、VARCHAR2 型に格納する。この変換は SQL ユーティリティ(SQLUtil)内部で行う。Boolean 型が True の場合 1 に、False の場合 0 に変換する仕様とした。また、RDB からの読み出しの後、String 型を Boolean 型に変換する。0 か 1 以外の値が RDB に格納されている場合には、本来ありえないこととして致命的エラーを返す仕様とした。

以上