顧客フレームワークとWebmentorの融合

金融システム
第二事業部
システム第二部
奈良祥生

金融システム
第二事業部
システム第二部
糸井　明

産業システム
第三事業部
CWeSソリューション
推進グループ
鳥海芳一

1．はじめに

インターネットの普及がもたらした「スピード化」の波は、ビジネス環境をも劇的に変化させた。システム開発においても短納期開発は常識となり、特にWebアプリケーションの構築では2、3ヵ月程度の開発期間も珍しくない。その一方で開発者に要求されるスキル・知識は広範囲に及び、従来のように一から全てを作り込むやり方では対応できなくなっている。このような状況で、コンポーネントやフレームワークの活用は今や必須と言っても過言ではない。

当社がWebシステム開発の新たなコンセプトとして提唱する「CWeS」*1にも、その中核となるフレームワーク製品として「Webmentor」*2が用意されている。また最近は、顧客サイドでも、他社製品や独自開発のフレームワークを標準採用しているケースも少なくない。本稿では、そのような顧客独自開発のフレームワークとWebmentorを併用した某金融機関向けJ2EEベースのWebアプリケーション開発事例を紹介する。

2．プロジェクトの背景

当初はWebmentor単独での開発として受注、設計を進めていたが、作成フェーズに着手しようかという段階で顧客独自開発フレームワークへの変更を打診された。今後の開発標準として、このフレームワークの使用を推進していきたいとの顧客意向もあり、現段階での変更可否を検討することとなった。検討を重ねた結果、開発標準としてのフレームワークの位置付けを最重要視し、Webmentor単独での開発方針を急遽変更、顧客フレームワークの使用を決定した。ただし、納期の変更はないため、フレームワーク変更によって発生する手戻り、追加作業を最低限に抑える方法を考える必要があった。その解決策として浮上したのが、顧客標準を遵守しながらWebmentorの高生産性を生かすフレームワーク融合方式である。

3．Webmentorの顧客FWへの適用詳細

3.1　顧客フレームワークの特徴

顧客フレームワーク（以下、顧客FW）の主な特徴としては、以下に挙げるとおりである（図1）。

(1) 顧客環境での標準フレームワーク

顧客が保有する業務アプリケーションの稼動環境には、サーバー構成ひとつをとってもWebサーバー、DBサーバー、さらにはAPサーバーといったように、さまざまな資源が存在する。そうした資源を最大限に有効活用するために提供されているのが顧客FWである。

(2) 統一的な規格の提供

顧客環境ではさまざまな業務アプリケーションが動作する。顧客FWを使用することで、顧客環境下にて複数稼動するアプリケーションが共存する上での統一的なルールを共有することができる。

(3) 機能の充実

顧客FWが提供する機能を使用することにより、帳票機

＊1）CWeSとはCAC Web Solutionの頭文字語。Web系システム開発のフレームを、システム基盤、業種･業務共有テンプレート、および顧客個別要件の三層に分け、再利用性・生産性・品質を高めている。

＊2）Webmentorは、CWeSの三層のうち、システム基盤のソリューションを提供する。

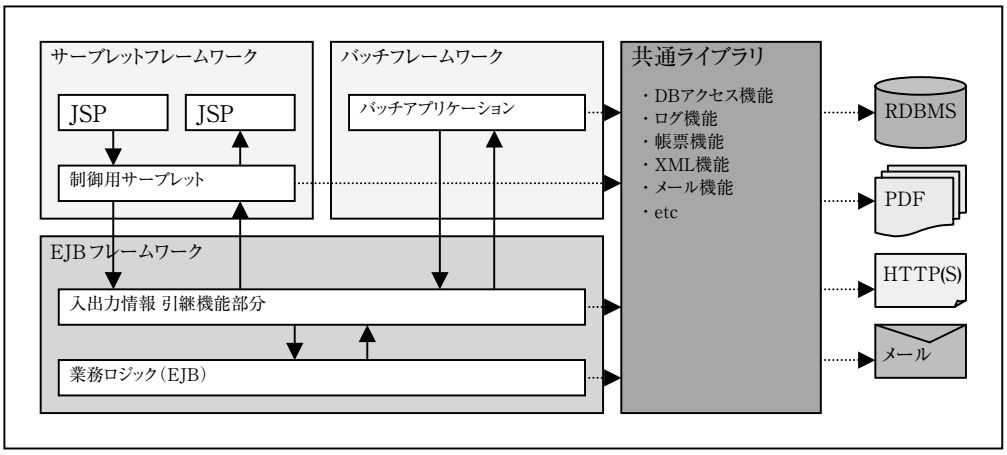

図1　顧客FW構成

能、メール機能、バッチ処理機能、HTTP（S）通信機能等が容易に実現可能である。

3.2　顧客FW使用に当たっての問題点

顧客FWの使用を前提とした開発計画を立てていく上で、次のような問題点が浮上した。

（1）開発部分の増加

顧客FW設計方針は開発効率よりも統一的なルールを与えることを重視しており、機能を実現させる上での制約も少なくない。提供される機能をそのまま使用できない場合は、開発者が一から作成する必要があり、開発規模が膨れる可能性がある。

（2）余裕のない開発期間

プロジェクト背景に述べたとおり、作成フェーズにさしかかってからの突然の方向転換により、リリースまでの時間的余裕が少ない。

（3）開発者に要求される高度なスキル

初めて使用する顧客FWの基盤となるアーキテクチャを理解した上でシステムの設計・製作を行うには、当初想定より高度な技術力が要求される。

以上の問題点はすべてがクリティカルなものであり、早急に解消することが求められた。

解決手段の一つとして、先に単独のフレームワークとしての採用が決定されていたWebmentorの利用を考察してみることにした。

3.3　Webmentorの特徴

Webmentorの特徴を以下に示す（次ページ図２）。

（1）洗練されたシステム設計

WebmentorはMVCモデル*3を採用しており、システムを機能レイヤごとに分割することで、コンポーネント開発が容易に実現できる。

（2）高い開発生産性

本来作成しなければならない、どのシステムにも共通する部分は、基底コンポーネントとして提供されており、J2EE/EJBなどの高度な知識を必要としない。このため、開発者は業務的なビジネスロジックの開発に集中でき、システム開発の生産性と品質を格段に高めることができる。

（3）外部製品に依存しない独立性

特定の製品、環境と結びつくことなく、汎用的に使用できるように体系づけられており、さまざまな外部部品との組み合わせが検討可能である。

3.4　Webmentor導入による効果

まず第一に、顧客FWを使用するには、アプリケーションが提供する各機能を処理していく上で必要なアーキテクチャ部分を実装することが必須であった。

たとえば、画面への入出力からビジネスロジックへの受渡しまでの処理部分においては、顧客FWでは情報引渡しクラスが提供されているのみである。このことは、JSP／HTML等から与えられたリクエスト内部より入力情報を引出す機能、入力情報を精査する機能などは、すべて開発者側で構築する必要があることを意味している。

例に挙げた一連の機能は、Webmentorにおいては、サーブレットにて該当箇所で利用できる情報引渡しクラスMessageCarrier、ビジネスロジックへ受渡す前の入力項目チェック等に利用できる事前処理クラス

＊3）オブジェクト指向言語SmallTalkでGUI設計の基礎概念として用いられたアーキテクチャ。WebアプリケーションはWebブラウザをGUIとして採用しているアプリケーションと見ることができるため、MVCモデルの概念を適用することが可能。「MVC」はそれぞれ「Model」「View」「Controller」を指し、J2EEになぞらえるなら「EJB」「JSP」「サーブレット」がそれぞれ当てはまる。

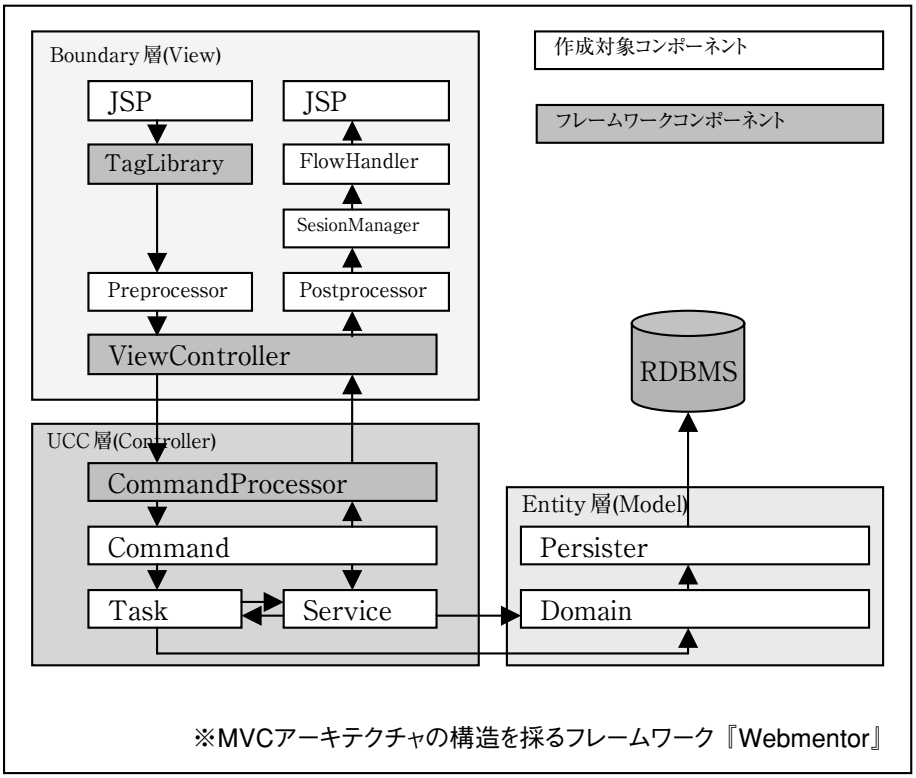

図2　Webmentor構成

Preprocessorにて実現可能である。

JSPからのリクエストにより呼び出されたサーブレットが、MessageCarrierを作成し、画面入力情報を格納させる。格納した情報は、Control層へ引渡し、ビジネスロジック処理される前にPreprocessorが呼ばれ、入力項目のチェックや整形等の処理を行い、ビジネスロジックにて正常に処理を遂行できる形へ操作する。

こういった顧客FWからは提供されていない機能をWebmentorから引出すことで、顧客FW内部をモデル化することが可能と考えられる。

モデル化を正しく行うことで、多岐にわたるユースケース処理でも共通した処理の流れを提供することが可能となり、冗長・重複したロジックの排除に役立つ。これにより該当部分の設計・製造にかかる人員、時間の削減につながり、大幅に工数を節約することも併せて期待できる。

また、本来あるフレームワークの特性として、フレームワーク外部へのアクセス簡素化、アプリケーション基礎部分のルール統一化、アプリケーション機能の拡張、他の部品との結合などによる少ない労力での要求機能の達成、などの点が挙げられる。

顧客ＦＷは、サービスの充実、環境全体での基本ルールの統一化・共有化といった部分の実現に重点が置かれている一方、環境全体の統一を目的としているためにフレームワークの部分部分を単体で用いるような使用方法は考慮されていない。

Webmentorはその設計方針自体が決まったアーキテクチャで構成されているため、同様の設計方針に従うなら、他の外部製品との組み合わせを検討することができる。

以上の点を考慮して、顧客ＦＷ内部でWebmentorを問題解決の手段として利用することを検討していった。

3.5　Webmentor導入により発生する問題点

本来、単独で使用するWebmentorを顧客ＦＷ内部で使用することにより、Webmentorが提供する基底コンポーネントを利用する上でいくつかの問題点が発生した。

(1) サーブレット

Webmentorを単独利用する場合には手を加える必要のないコンポーネントの一つとしてServletクラスが挙げられる。

今回、顧客FWを利用する上で、サーブレットは顧客FWのベースクラスを継承することが義務付けられているため、該当クラスに関しては独自作成をする必要がある。

しかし、顧客ＦＷのＳｅｒｖｌｅｔクラスを継承し、WebmentorのServletクラスを利用しないために、使用できない機能が発生した。

Ｗｅｂｍｅｎｔｏｒでは、各画面への入力情報、遷移情報、サーブレット以降で使用するクラス指定などをXMLにより構成された「画面定義ファイル」にて設定する。

これはViewController（Webmentorが提供するServletクラス）より利用されるものであるため、顧客FWが提供

するServletクラスからでは当然利用できない。

(2) EJB関連クラス

サーブレットと同様、EJB部分もWebmentor単体で利用する場合は、開発者側で手を入れる必要はない。

しかし今回、顧客FWのベースクラスを継承したEJBコンポーネントを必ず利用しなければならない点もサーブレットと同様である。

よって、EJB関連の機能に関しては顧客FWのものを使用するしかなく、Webmentorが備えているEJB部分の使用は諦めざるを得なくなった。

WebmentorのEJBクラスが使用できない事実は、同コントローラ層に位置するCommandクラスを呼び出すロジックが利用できないという問題点を浮上させた。

(3) DBアクセス

DBへのアクセスを行う場合、Webmentorの機能としてはJDBCPersisterコンポーネントが既に提供されている。

JDBCを用いてオブジェクトの永続化等を図るには、上記クラスを利用することで可能となる。

しかし、顧客FWの方針として、DBへのアクセスを行う場合は必ず顧客FW提供のDBアクセスクラスを使用するよう、規約にて定められている。

モデル層においても、Webmentorの機能に制限を受けざるを得ない事態となった。

(4) 例外処理

Webmentorおよび顧客FWとも独自の例外処理を規定している。両者とも発生した例外イベントに対し利用者により多くの情報を与えられるよう、通常のJ2SEにて提供されている基底の例外クラスExceptionに拡張を行い、独自の例外クラスを形成している。　顧客FWにおいては、EJB呼出し部分、入出力情報の引継ぎ部分、サーブレットの3点において例外をキャッチし、例外内容に対して個別の処理を行っている。

しかし、そこで受け入れられるのは、同じく顧客FWが提供する例外クラスのみで、その他の例外クラスは予期されない例外として処理される体系が採られている。

顧客ＦW内部にてWebmentorの提供するクラスを使用する場合、そこから発生する例外はWebmentor独自の例外クラスであり、両フレームワーク間に齟齬が生じる結果となってしまう。

(5) Preprocesssor（事前処理）クラス

まず、WebmentorではJDBCを利用するにはUCC層を介する必要がある。

加えて、Webmentorが提供するPreprocesssorクラス（事前処理クラス）、Postprocessorクラス（事後処理クラス）などは、それぞれEJBにてビジネスロジックが処理される前後で実行されるものである。

開発者側の要望として、EJBでの処理に入る前に、入力項目のチェック等の処理に際して既にRDBMSにて永続化されている情報を利用したい場合がかなりの頻度で発生することが予想できた。

しかしWebmentorが用いるMVCアーキテクチャの方針では、Boundary層に位置するプリプロセッサ等が、Entity層にあるDBアクセスクラスを使用することは、設計方針上、許可されることではない。

3.6 問題点の解決

コンポーネントごとの解決方法を以下に説明する。

(1) サーブレット

Webmentorの提供するServletクラスを継承しないことにより、固有の提供機能が使用できなくなったが、この点に関してはWebmentor内部にて処理されている部分であり、Webmentor利用者側では解決できない問題であった。

しかし、この問題点についてはWebmentor開発チームからの提供により、他製品のクラス上からでもWebmentorの機能が使用できるよう、追加ユーティリティクラスの提供というカスタマイズ方法で対応が可能となった。

提供されたユーティリティクラスを利用することで、顧客FWのServletクラスを継承しつつ、Webmentorが備える機能をも利用可能な体制を整えることができた。

以下に、Webmentorのユーティリティクラスを取り入れた後のアプリケーション体系図およびメッセージの遷移を表すシーケンス図を示す（図３、４）。

(2) EJB関連クラス

顧客FWを使用する上で、EJB部分の実装は顧客FWのクラスを継承することが必須である。よって、WebmentorのEJB関連クラス（Commandプロセッサ等）を使用できない。

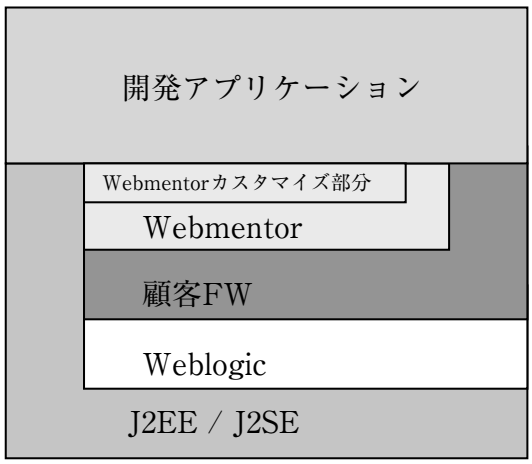

図3　Webmentorユーティリティクラスを導入した当プロジェクトのアプリケーション体系

顧客FWのEJB呼出し方法は、サーブレットよりEJB以下ビジネスロジックへ画面入力値等の情報引継ぎ機能を有したクラスがEJBの生成・実行を行う、といったものである（図5）。

Webmentorでは、EJB以下ビジネスロジックを実行するに際して、CommandProcessorからCommandクラスを呼出すことで処理を行っている。

この処理の流れを顧客FWでも実現するために、以下のような方法を採った。図6のシーケンス図を参照してほしい。

・Servletクラスにて Commandクラスを生成し、画面入力値などの情報を保持させる。
・Commandクラス全体を顧客FWの情報引継ぎクラスへ保持する。
・情報引継ぎクラスがEJBを生成、実行を行う。
・実行されたEJBに情報引継ぎクラスが保持するCommandクラスを取得させる。
・EJBロジック内にて、Commandクラスを実行する。

以上の流れによって、ServletクラスからEJBを介しCommandクラスを実行することを可能とした。

Commandクラスからは、更に詳細なビジネスロジック実装部分としてTaskクラスが実行される。Taskクラスは情報の永続化等を行うEntity層のクラスにアクセスし得る唯一のクラスである。

このようにして、Boundary層からUCC層を介し、Entity層へ入力情報を引き渡すための一本化した処理体系を見出すことに成功した。

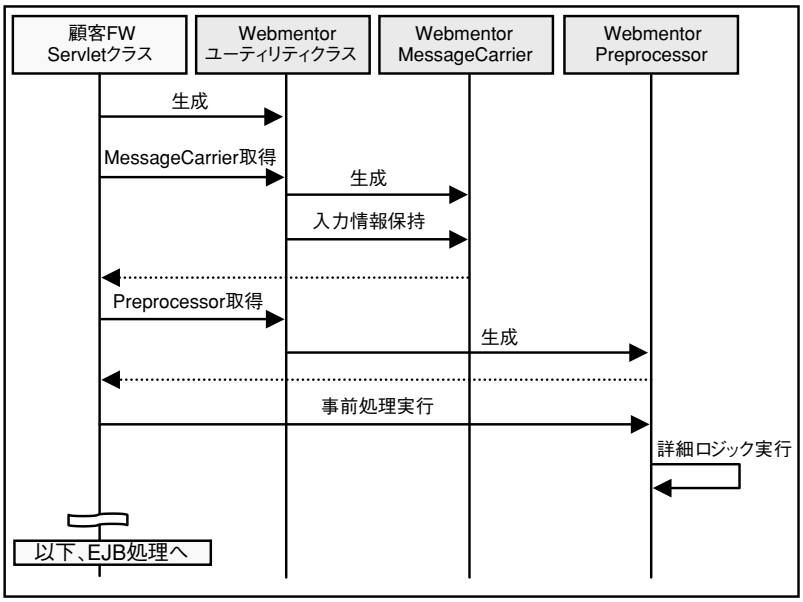

図4　Webmentorユーティリティクラスを介した提供クラスのコール

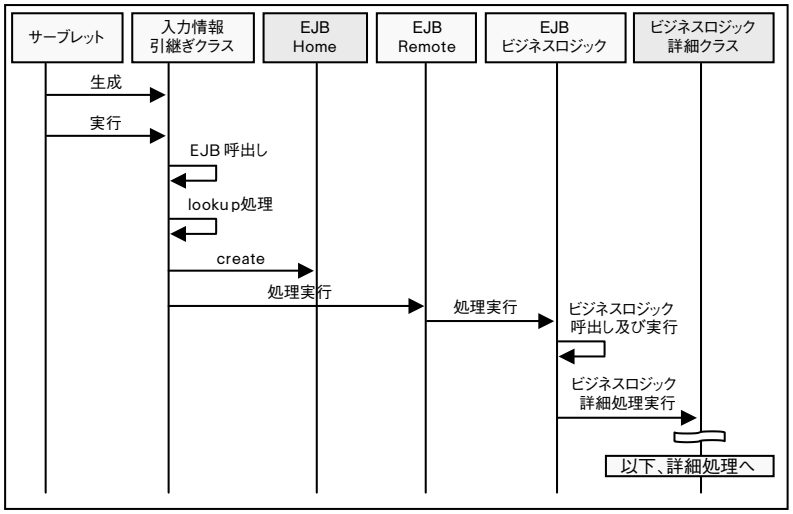

図5　顧客FWのEJB処理パターン

（3）DBアクセス

顧客FWを使用する上では、DBアクセスに際しても顧客FWのクラスを使用することが義務付けられている。

Webmentorをそのまま使用できれば、DBアクセスを行うPersisterクラスを用いれば良い。

PersisterクラスにはDBへの検索・更新を行うメソッドが既に実装されているため、アプリケーション開発者は基本的には手を加える必要はない。

顧客FWを使用する場合はDBアクセスクラスを使用するわけだが、DBへのアクセスを行う手段として、特定のメソッドにSQL文をそのまま引数として渡すという最低限のメソッドしか実装されていない。

従って、当アプリケーションではWebmentorでのPersisterクラスと同じ位置付けのクラスを独自作成し、DBへの検索・更新を行うようにした。

作成したクラスの実装メソッドには顧客FWのDBアクセスクラスを使用するようにロジックを組み、実装メソッドには最低限の情報だけを引数として渡すことでDBアクセスを容易に実行可能とした（図7）。

（4）例外処理

顧客FWには、発生した例外を処理する場所として必ず通過する部分が、大きく分けて3ヵ所ある。

該当箇所へ、顧客FWにて提供されているクラス以外の例外を受け渡してしまうと、それがたとえ意図的に発生さ

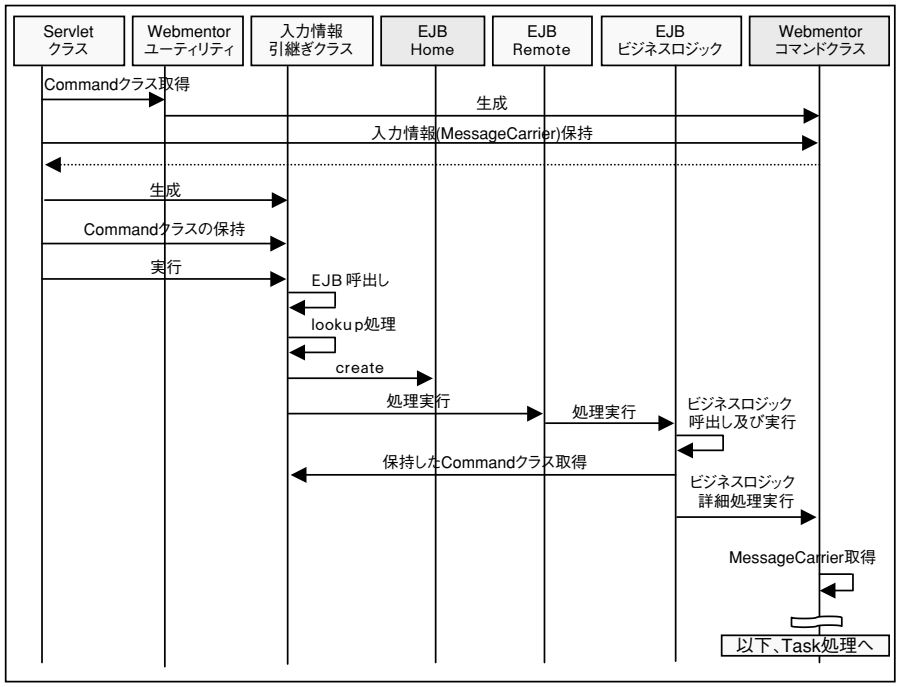

図6　Webmentorのビジネスロジックを呼出す顧客FWのEJB処理パターン

せたものであっても、そのすべてが予期せぬ例外として扱われてしまう。予期せぬ例外は、アプリケーションの処理に対しアベンド終了を引き起こさせる。

ここまで述べたとおり、当アプリケーションのビジネスロジック等ではWebmentorのクラスを使用するため、通常の処理において発生する例外はWebmentorの例外クラスとなる。

両フレームワークが提供する例外クラスは、ともにJ2SEの基底例外クラスである。しかし、Webmentorの例外クラスは、顧客FW下では顧客FW内の例外クラスとして認識されない例外クラスと判断され、予期せぬ例外が処理中に発生したものとして例外を処理しようとする。

両者間に生じるこの齟齬を解消する方法として、顧客FWにて、例外を処理する箇所を通過する前に発生している例外クラスを、顧客FWの例外クラス内に保持するよう、ハンドル処理を行うようにした（次ページ図８）。

（5）事前処理 Preprocesssorクラス

これに関しては、問題点というより開発者側からの要望に近いものだったが、1）のServletクラスの場合と同様に、PreprocesssorクラスでもJDBC参照が可能となるよう、Webmentorユーティリティクラスへの機能提供の形で実現することができた。

具体的な変更点は、ServletクラスよりPreprocesssorを生成する際にコールするユーティリティクラスのメソッドへ渡す引数に、EJB呼出しクラスを加えたことだ。

この変更により、顧客FWにてEJB呼出しに利用される情報引継ぎクラスをPreprocesssorクラスは保持できるようになり、生成時に顧客FWの情報引継ぎクラスを保持させた状態のPreprocessorクラスがサーブレットから参照できるようになった（次ページ図９）。

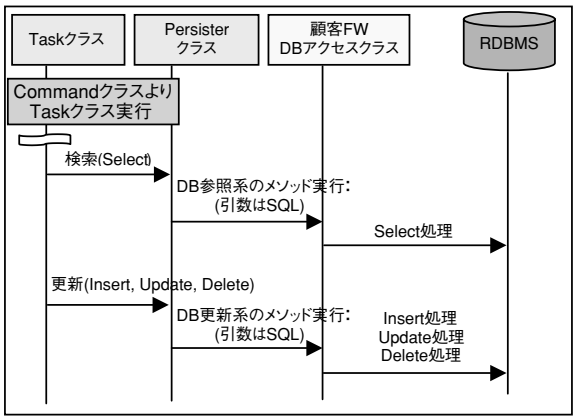

図7　顧客FWのDBアクセスクラスを使用したPersisterクラスの動作

ちなみに、PosrprocessorクラスはEJB呼出し後に呼ばれるクラスであり、DBの情報を参照したい場合は前もってビジネスロジック内での取得が可能であるため、仕様変更の対象とはならなかった。

3.7　顧客FWとWebmentorの最終的な配置

先に挙げた問題点を解決した上で、２つのフレームワークを融合させた最終的な構成は43ページ図10のようになった。

全体的なベースは顧客FWのクラスにて構成されてお

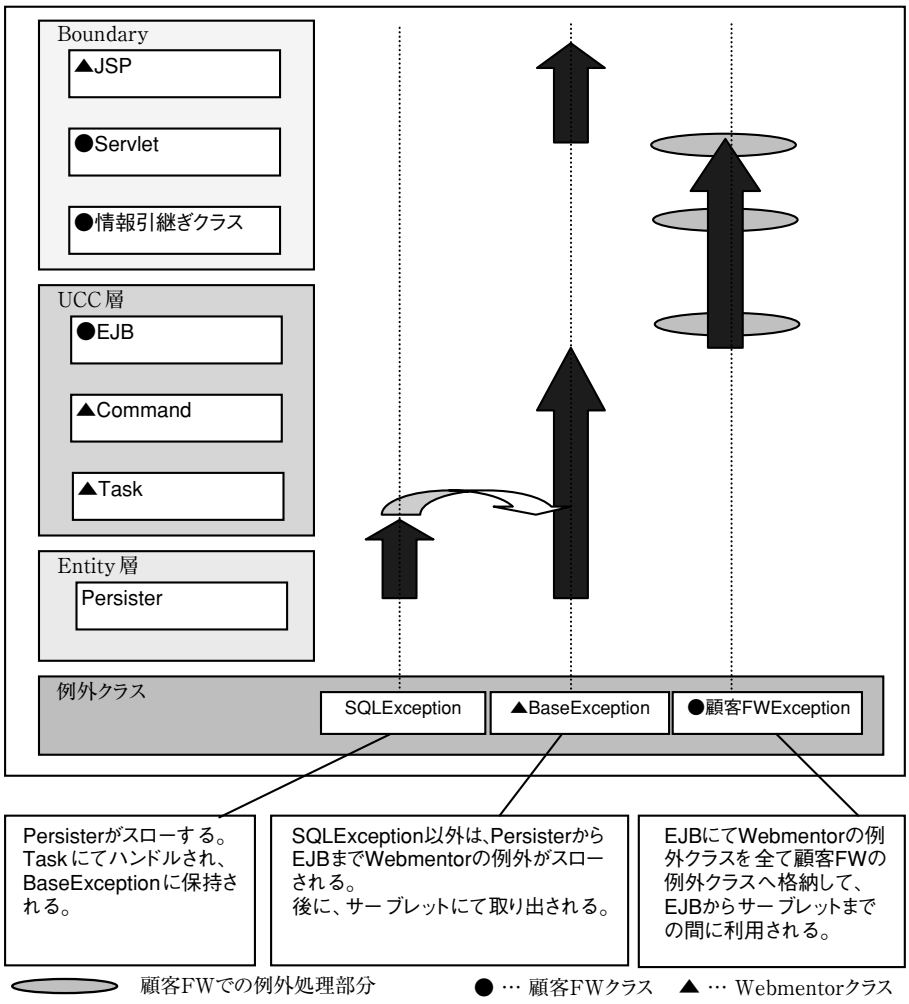

図8　例外クラスの処理方法

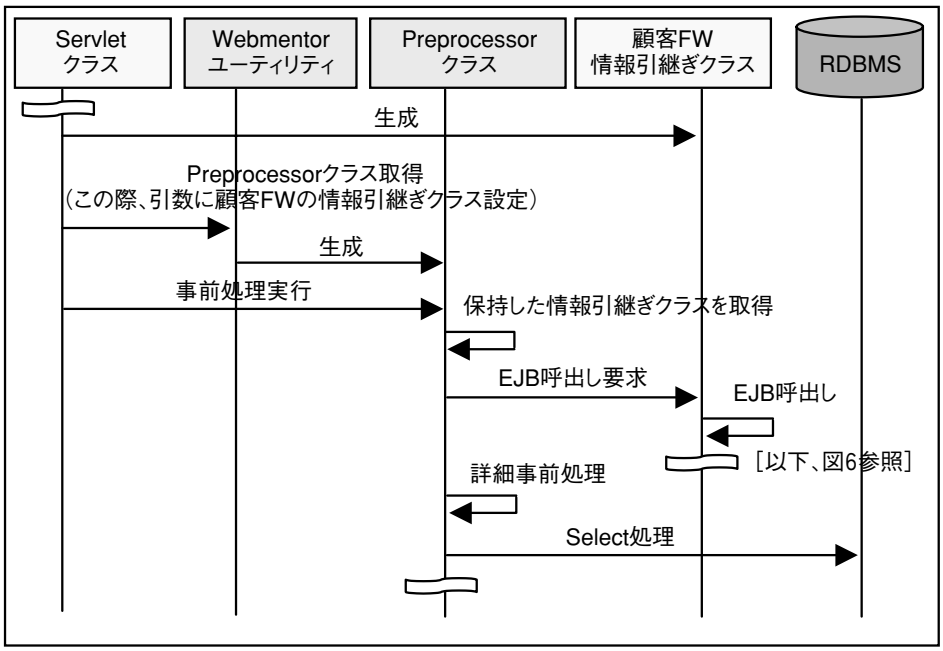

図9　PreprocessorクラスでのJDBC利用

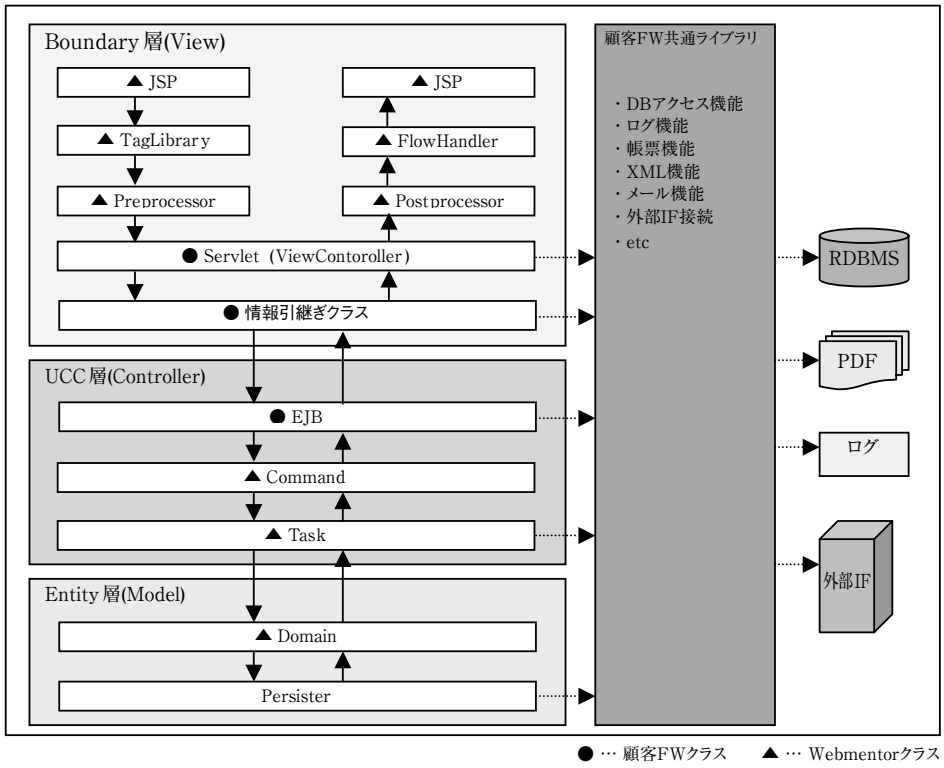

図10　最終的な処理体系

り、また最上位クラスでは顧客FWのモジュールを利用している。しかし、設計方針はWebmentorのアーキテクチャと似た形で構成されているのが理解できるだろう。

4．最後に

今回の２つのフレームワークの融合は初めから意図したものではないし、すべての事例で同様の方式が可能というわけではない。当然のことながら、Webmentorを使用しないという選択肢があったのも事実であり、セオリーから外れるという意見もあるかと思う。ただ、当事例では、大きく次の２つの点で満足のいく結果となった。

１つ目は開発効率。今回のプロジェクトにはWebアプリケーション構築が初めてのメンバーも数人いたが、開発後半には、決して小さくない規模のアプリケーションをかなりの短期間で構築していた。Webmentorの高い生産性を有効に利用できたものと考える。

２つ目は成果物の品質。開発途中の大幅変更にもかかわらず、機能や納期は当初のプランどおり実現し、大きなトラブルもなくパフォーマンスも含め安定稼動している。結果として、同一プラットフォームでの別案件受注にも繋がり、現在も継続している状況である。

Webmentor単独での使用以外にも、必要な機能だけに絞った使用法（各プロジェクトでのカスタマイズを前提として、ソースコードベースでの提供）も、今後の選択肢の１つになるだろう。