アストラル・アパッチ
開発ガイド

アストラル・アパッチの各種ファイル説明

1. 設定ファイル

設定ファイルは Tomcat の web.xml と開発したアプリケーションで使用する設定ファイルがある。Tomcat の web.xml は Tomcat バージョン名¥webapps¥アプリケーション名¥WEB-INF の下にある。

開発したアプリケーションで使用している３つの設定ファイルは Tomcat バージョン名¥webapps¥アプリケーション名の下に設置する。各ファイルの説明は表１、表２を参照すること。

表１ TOMCAT で使用する設定ファイル

ファイル名	機能
web.xml	Tomcat の定義ファイル。Servlet 等を定義する。

表２ アプリケーションで使用する設定ファイル

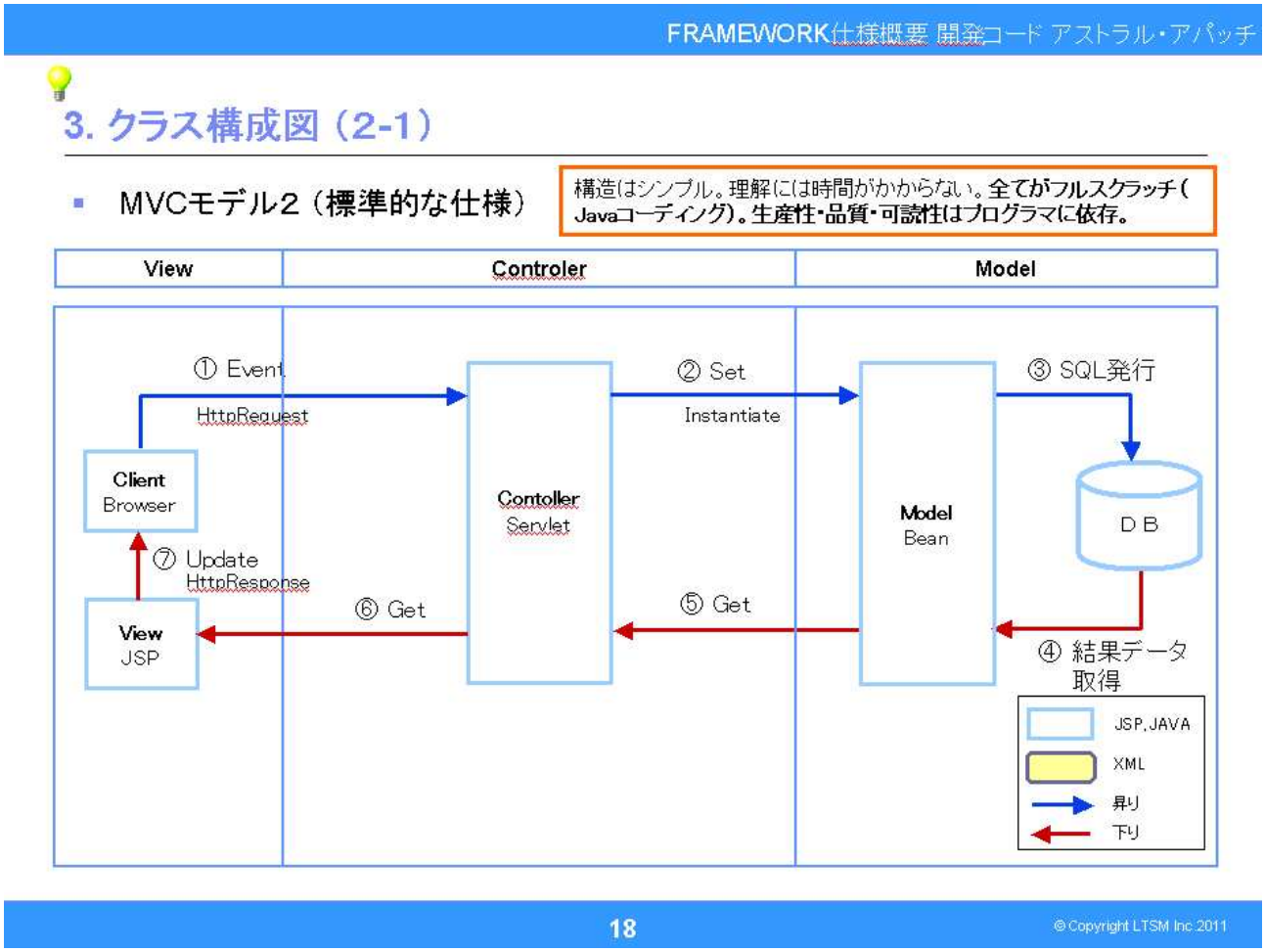
ファイル名	タイプ	機能	命名規則
actionmap.xml	仮想 URL の定義	HTTP リクエストに呼び出される機能（クラス／メソッド）を仮想 URL にする定義を記述する。	固定
resultmap.xml	jsp の仮想名称の定義	Jsp の仮想呼出名称の定義を記述する。	固定
message.xml	廃止(現行バージョンでは廃止)	エラーメッセージを登録する。	廃止
Action.xml	制御用 XML 定義	処理の制御を記述する。具体的には、画面のボタン押下などの情報を取得し、ボタン押下時の処理の実態を記述する Event クラスの指定メソッドを呼び出す。	機能名+Action.xml
Data.xml	画面等項目の XML 定義	画面項目名、レングス、属性、各種入力チェック、データベースとのマッピングなどの画面等項目に関する情報を記述する。	機能名+Data.xml
Form.xml	画面で使用するデータクラスの XML 定義	画面上で使用するデータクラスを記述する。１画面に１つ作成する。Form.java に継承させる。	機能名+Form.xml
Form.java	Java Class	SP へのデータ値受け渡し用のクラス。Form.xml を継承し、１画面に１つ作成する。	画面名Form.java

2. クラス構成図

【WEB サービスの本来要件　Java のケース】
Java プログラマーは HTML ではなくサービスの開発をすべき。
レイアウトを変更するたびにコードを変更する必要がない。
サービスの利用者が、それぞれ特有の必要に従ってページを作成することが可能でなければならない。
ページ・デザイナーが、ページの開発に直接関係することができる。
コード中に HTML を組み込むのは、見苦しい方法である。

上記のような前提条件を具備するために提案された手法がＭＶＣモデル２である。

標準的なＭＶＣモデル２



【ＭＶＣモデル２の概要】

JSP ＋ Servlet ＋ Beans（JavaClass） で構成

ブラウザからのリクエスト受付と、各オブジェクト間の制御を行うコントローラの役割は、Servlet が担う。

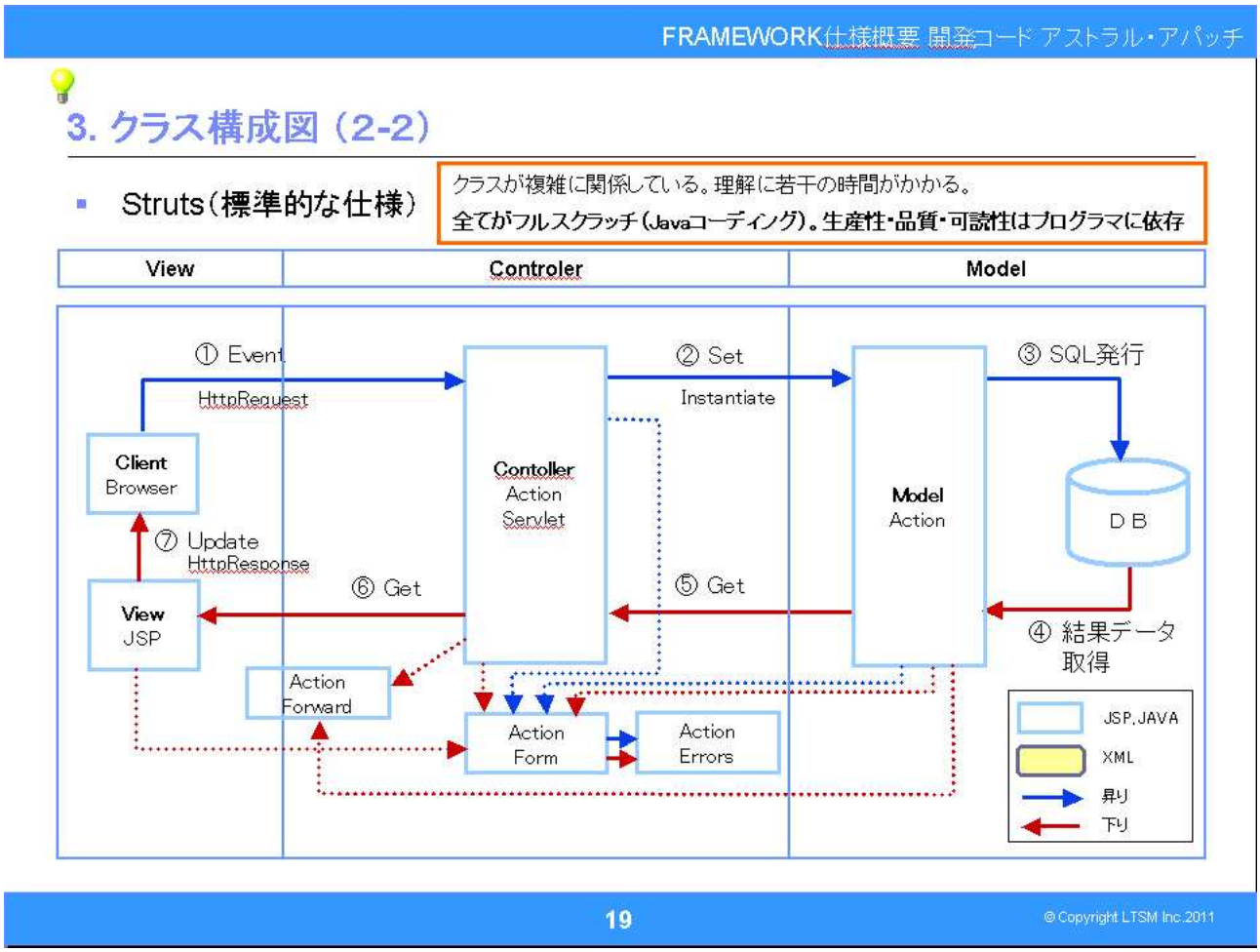
【メリット】

コントローラと、画面デザインと、業務ロジックを分離しているため、再利用性と保守性に優れたモデルである。構造はシンプル。理解には時間がかからない。

【デメリット】

全てがフルスクラッチ（Java コーディング）。生産性・品質・可読性はプログラマに依存。

標準的な Struts



Struts はＭＶＣモデル２の自由度をより制約することで、プログラマ毎に異なるコードが作成される事防ぎ、可読性と再利用性をより高める事を目的に提案された。

【Struts の概要】
クライアント・ブラウザー
クライアント・ブラウザーからの HTTP 要求により、イベントが作成されます。Web コンテナーは、HTTP 応答を戻します。

コントローラー
コントローラーは、ブラウザーからの要求を受け取り、どこにその要求を送るべきかを決定します。Struts の場合、コントローラーは、サーブレットとして実装されたコマンド・デザイン・パターンです。コントローラーの設定には struts-config.xml ファイルが使用されます。

ビジネス・ロジック

ビジネス・ロジックは、モデルの状態を更新し、アプリケーションのフロー制御を支援します。Struts の場合、このことは、実際のビジネス・ロジックに対するラッパーとして Action クラスを使用してなされます。

モデル状態
モデルは、アプリケーションの状態を表しています。アプリケーションの状態は、ビジネス・オブジェクトによって更新されます。ActionForm bean は、永続的レベルではなく、セッション・レベルまたは要求レベルでのモデル状態を表します JSP ファイルは、JSP タグを使うことによって、 ActionForm bean から情報を読みます。

ビュー
ビューは、単なる JSP ファイルです。フロー・ロジック、ビジネス・ロジック、モデル情報は含まれず、タグが含まれるだけです。Velocity などのフレームワークと比較した場合に、タグは Struts を特徴付けるものの１つです。

【Struts のメリット】
ＭＶＣモデル２より、再利用性が高い。

【Struts のデメリット】
クラスが複雑に関係している。理解に若干の時間がかかる。
全てがフルスクラッチ（Java コーディング）。生産性・品質・可読性はプログラマに依存。

アストラル・アパッチとは

【アストラル・アパッチの概要】

【JAVA プログラムを減らす】

システムの記述言語に XML を採用し、JAVA プログラムの量を減らす事で、モレ、ミスなどのバグを減らし、品質と可読性を向上し、生産性（開発速度）を上げる事を実現した。このフレームが採用している概念は MDA(Model Driven Archtecture)である。XML 70% java 30% 程度のプログラム比率になっている。

【設計書とプログラムを一致させる】

オープン系システム開発では、設計書とプログラムは一致しない。設計・製造期間が短い、仕様変更時に設計書を直さない、スキル不足の技術者が多いことなどの原因が上げられるが、期間やスキルの問題を今すぐに改善できるものではない。よって、スキルではなく、技術により、設計書とプログラムを同期させる機能を実現した。(独自の MDA(Model Driven Archtecture)エンジンを開発。)

【お客様にとって重要な事は結果である】

お客様にとって最も重要な事は３つ、予算内、期間内、求める品質の実現だけである。つまりどのような設計方法、製造方法、実現の方式かは重要ではなく、求める業務に沿った機能、予算内、期間内、求める品質を実現することだけが重要である。

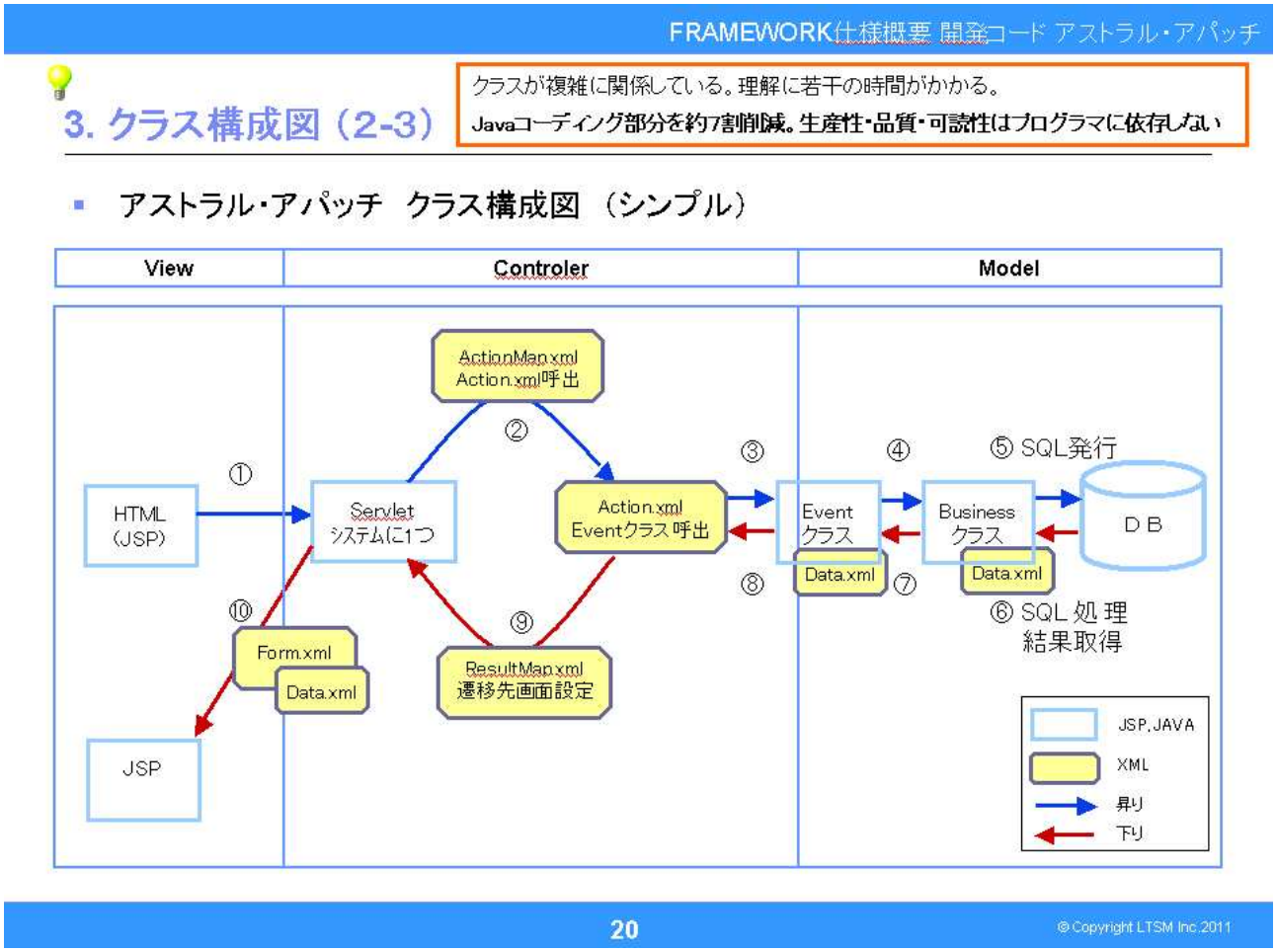
アストラル・アパッチは、上記の予算内、期間内、求める品質の実現と利益の最大化に最適化されている。つまり、SI の技術指向を無視し、お客様利益を最大化するフレームワークである。(SI の技術指向とは Java、Struts、EJB、その他現在までに発生した様々な言語や複雑な技術を指す。システムとは、お客様の目的と結果が一致すれば、どのような実現方法でも構わないというのが弊社の考えで、本来は COBOL のような生産工学に基づく言語こそ素晴らしいという考えである。)

画面、項目、データベースとのマッピングなどの設計情報を XML で記述することで、アストラルアパッチが設計情報を読み込みシステム稼動する。また XML 設計情報から、各種設計書を自動作成する。

【アストラルアパッチの特徴】

設計情報を XML で記述することで、アストラルアパッチが設計情報を読み込みシステム稼動する。このため、基本設計フェーズからシステムのプロトタイプを稼動させることが可能。
項目の仕様変更（必須入力、レングス、属性など）は XML 定義を変更するだけで済む。つまりコンパイルは必要ない。
項目の追加、削除は XML 定義の変更と、わずかな Java プログラムの修正で完了する。
新規画面の開発は、用意された雛形の指定文字を一括置換するだけで、画面の基本的な動作を実現可能。
設計情報を記述した XML から、各種設計書を出力できる。

アストラル・アパッチ　標準的な構成

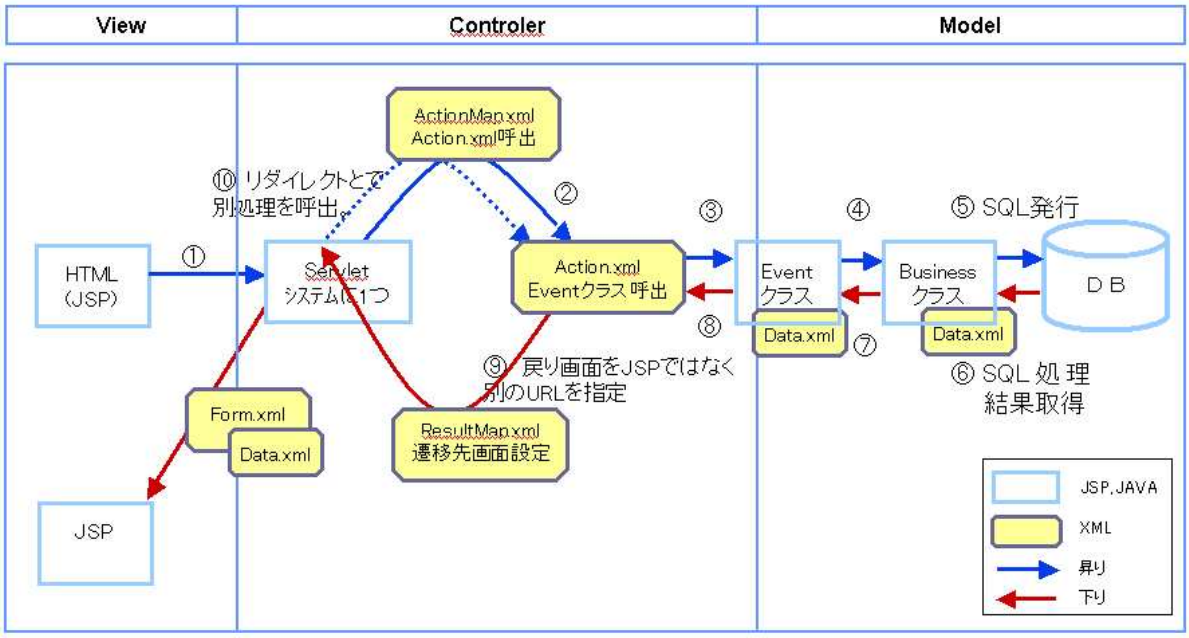


【アストラル・アパッチの標準的な構成】

アストラル・アパッチ　リダイレクトで再度別処理呼出

3. クラス構成図（2-4）

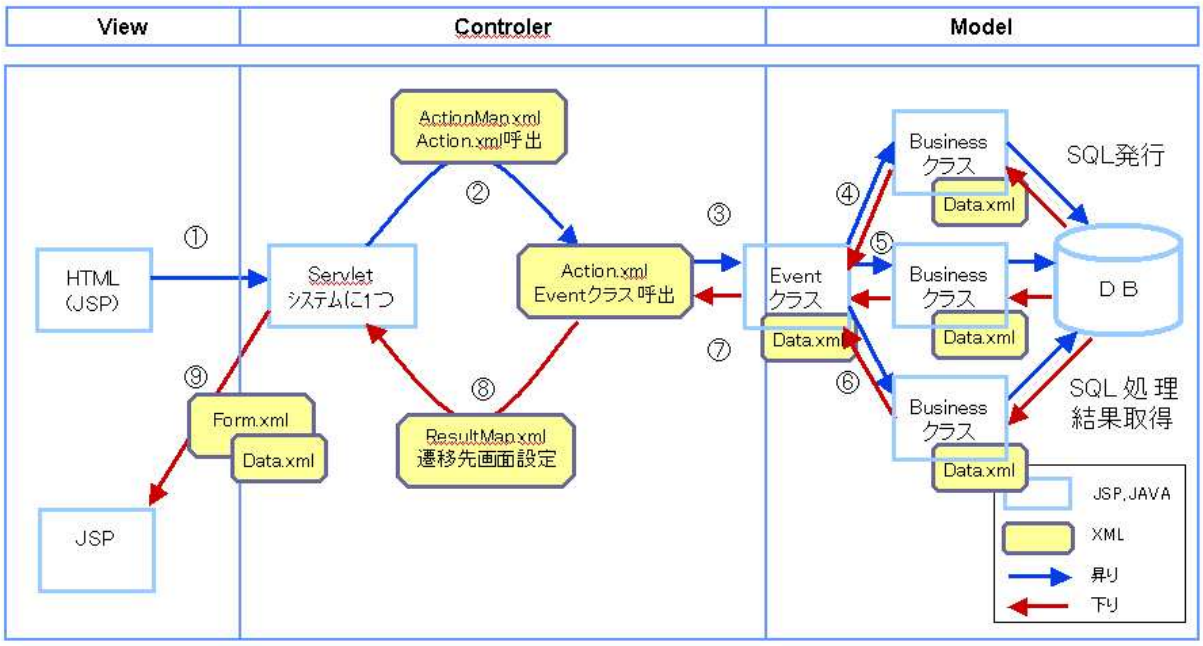
- アストラル・アパッチ　クラス構成図（リダイレクトで再度別処理呼出）



アストラル・アパッチ　イベントクラスから複数のビジネスクラスの呼出

3. クラス構成図（2-5）

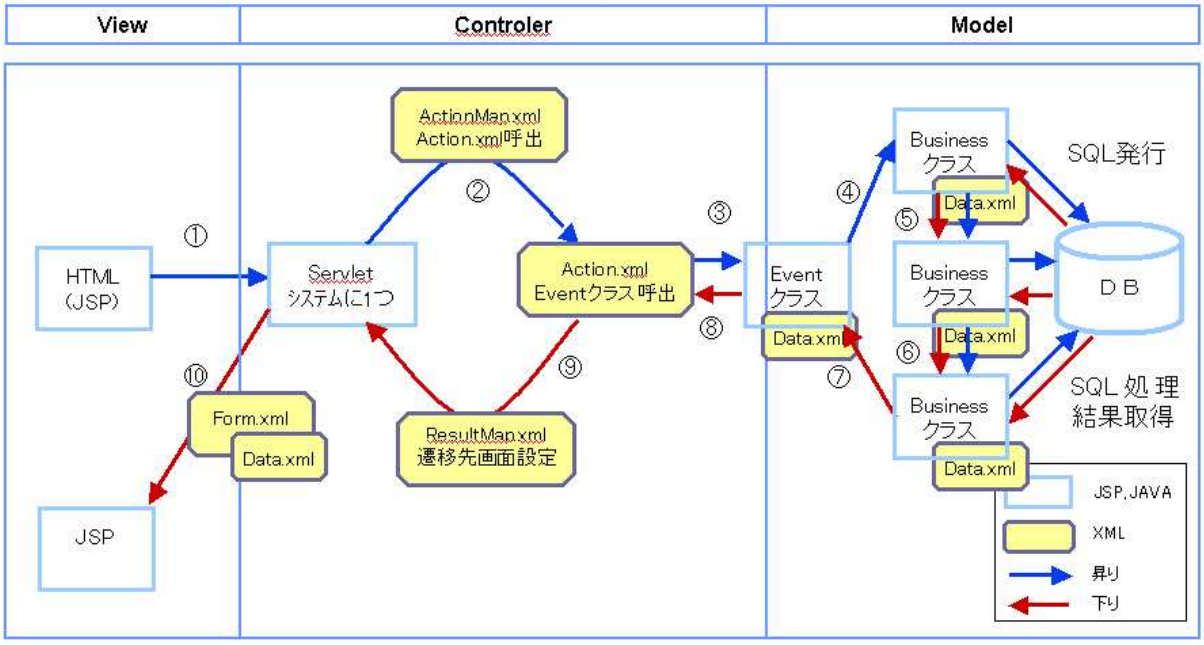
- アストラル・アパッチ　クラス構成図（イベントクラスから複数のビジネスクラスの呼出）



アストラル・アパッチ　ビジネスクラスから別のビジネスクラスの呼出

3. クラス構成図（2-6）

- アストラル・アパッチ　クラス構成図（ビジネスクラスから別のビジネスクラスの呼出）



3. web.xml の設定内容

```
<?xml version="1.0" encoding="UTF-8"?>

・・・中略

<web-app>

        <context-param>
                <param-name>web_app_name</param-name>
                <param-value>todo</param-value>
        </context-param>
        <context-param>
                <param-name>driver</param-name>
                <param-value>oracle.jdbc.driver.OracleDriver</param-value>
        </context-param>
        <context-param>
                <param-name>url</param-name>
                <param-value>jdbc:oracle:thin:@ 127.0.0.1:1521:ORCL </param-value>
        </context-param>
        <context-param>
                <param-name>db_user</param-name>
                <param-value> sample </param-value>
        </context-param>
        <context-param>
                <param-name>db_password</param-name>
                <param-value> oracle </param-value>
        </context-param>
        <context-param>
                <param-name>max_connection</param-name>
                <param-value>10</param-value>
        </context-param>
        <context-param>
                <param-name>auto_commit</param-name>
                <param-value>false</param-value>
        </context-param>
        <context-param>
                <param-name>auto_check_screen_param</param-name>
                <param-value>true</param-value>
        </context-param>
        <context-param>
                <param-name>debug_mode</param-name>
                <param-value>true</param-value>
        </context-param>
        <!-- synch_mode:xls, no -->
        <context-param>
                <param-name>synch_mode</param-name>
                <param-value>no</param-value>
        </context-param>
        <context-param>
                <param-name>tomcat_a_url</param-name>
                <param-value>http://127.0.0.1:28080</param-value>
        </context-param>
        <context-param>
                <param-name>tomcat_b_url</param-name>
                <param-value>http://127.0.0.1:28080</param-value>
        </context-param>


・・・中略
```

コンテキスト名

JDBC Driver

JDBC の接続文字列

ORACLE のユーザー

ORACLE のパスワード

コネクションプールの最大プール数

auto_commit の設定

check_screen_param の使用有無

debug_mode 使用有無
printDebug メソッドを使用したデバッグ用出力
true：コンソールに出力
false：出力なし

synch_mode の使用有無
xls：設計書出力
no：通常モード

web.xml の設定内容

```
・・・中略
        <servlet>                                          Servlet の設定
                <servlet-name>CommonControl</servlet-name>
                <servlet-class>common.CommonControlServlet</servlet-class>
        </servlet>
        <servlet>
                <servlet-name>LoginControl</servlet-name>
                <servlet-class>todo.LoginControlServlet</servlet-class>
        </servlet>

        <servlet>
                <servlet-name>TodoControl</servlet-name>
                <servlet-class>todo.ControlServlet</servlet-class>
        </servlet>

        <servlet-mapping>                                  Servlet-Mapping の設定
                <servlet-name>LoginControl</servlet-name>
                <url-pattern>/LoginControl/*</url-pattern>
        </servlet-mapping>

        <servlet-mapping>
                <servlet-name>TodoControl</servlet-name>
                <url-pattern>/TodoControl/*</url-pattern>
        </servlet-mapping>

        <session-config>                                   session-timeout の設定
                <session-timeout>120</session-timeout>     分単位で設定する。
        </session-config>                                  <!-- 120 minutes -->

        <taglib>                                           taglib の設定
                <taglib-uri>common-taglib.tld</taglib-uri>
                <taglib-location>/WEB-INF/common-taglib.tld</taglib-location>
        </taglib>
</web-app>
```

4. actionmap.xml の設定内容

```
<?xml version="1.0" encoding="Shift_JIS"?>
<action>

        <actionmap>

                <!-- ログイン画面 -->
                  <pathinfo>/login</pathinfo>
                  <actionhandle>todo.action.LoginAction</actionhandle>

                <!-- TODO 一覧画面 -->
                  <pathinfo>/todo_list</pathinfo>
                  <actionhandle>todo.action.TodoListAction</actionhandle>

                <!-- TODO 明細画面 -->
                  <pathinfo>/todo_detail</pathinfo>
                  <actionhandle>todo.action.TodoDetailAction</actionhandle>

        </actionmap>
</action>
```

アクションクラスを呼出す為のパス

実際のクラス

5. resultmap.xml の設定内容

```
<?xml version="1.0" encoding="Shift_JIS"?>
<action-result>
        <resultmap>

                <!-- ログイン画面 -->
                <result>login</result>
                <jsppath>/jsp/Login.jsp</jsppath>

              <!-- TODO 一覧画面 -->
                <result>todo_list</result>
                <jsppath>/jsp/TodoList.jsp</jsppath>

              <!-- TODO 一覧画面へリダイレクト -->
                <result>redirect_todo_list</result>
                <jsppath>/TodoControl/todo_list</jsppath>

              <!-- TODO 明細画面 -->
                <result>todo_detail</result>
                <jsppath>/jsp/TodoDetail.jsp</jsppath>

              <!-- TODO 明細画面へリダイレクト -->
                <result>redirect_todo_detail</result>
                <jsppath>/TodoControl/todo_detail</jsppath>

                <!-- 削除等確認フォーム -->
                <result>confirm</result>
                <jsppath>/specific/Confirm.jsp</jsppath>

        </resultmap>
</action-result>
```

VIEW(JSP)を呼出す為のパス

実際の JSP

6.　message.xml の設定内容

```
<?xml version="1.0" encoding="Shift_JIS"?>
<message>
        <messagemap>

                <messageno>YOU001</messageno>
                <prompt>入力値に誤りがあります。</prompt>

                <messageno>YOU002</messageno>
                <prompt>入力必須項目です。</prompt>

                <messageno>YOU003</messageno>
                <prompt>無効キーです。</prompt>

                <messageno>YOU004</messageno>
                <prompt>該当するデータはありません。</prompt>

                <messageno>YOU005</messageno>
                <prompt>ファイルのＩ／Ｏエラーです。</prompt>

                <messageno>YOU006</messageno>
                <prompt>該当コードがファイルに存在しません。</prompt>

                <messageno>YOU008</messageno>
                <prompt>この項目には、入力できません。</prompt>

                <messageno>YOU009</messageno>
                <prompt>この欄は、入力が必要です。</prompt>

                <messageno>YOU010</messageno>
                <prompt>すでに登録済みのコードです。</prompt>

                <messageno>YOU011</messageno>
                <prompt>前頁はありません。</prompt>

                <messageno>YOU012</messageno>
                <prompt>次頁はありません。</prompt>

                <messageno>YOU013</messageno>
                <prompt>企業マスタに未登録です。</prompt>

                <messageno>YOU014</messageno>
                <prompt>入力した日付は実在しません。</prompt>

                <messageno>YOU015</messageno>
                <prompt>入力した日付は未来日付です。</prompt>

                <messageno>YOU016</messageno>
                <prompt>入力した日付は過去日付です。</prompt>

                <messageno>YOU017</messageno>
                <prompt>範囲指定大小比較エラーです。</prompt>

                <messageno>YOU108</messageno>
                <prompt>○×△ 例）項目には、入力できません。</prompt>

・・・中略

        </messagemap>

</message>
```

メッセージ ID

実際のメッセージ

7. Action.xml の設定内容

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<action>
        <form id="todoDetailForm" scope="session" autoSetScreenParam="true">
                todo.form.TodoDetailForm
        </form>

        <!-- デフォルト処理 -->
        <isClicked>
                <comment>デフォルト処理</comment>
                <event method="defaultEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

        <!-- 参照ボタン押下 -->
        <isClicked>
                <comment>参照ボタン押下</comment>
                <pushButton>btnRefer</pushButton>
                <event method="referEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

        <!-- 新規作成ボタン押下(新規作成画面に遷移) -->
        <isClicked>
                <pushButton>btnReferInsert</pushButton>
                <event method="referInsertEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

        <!-- 新規作成ボタン押下 -->
        <isClicked>
                <pushButton>btnCommitInsert</pushButton>
                <event method="commitInsertEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

        <!-- 更新ボタン押下(更新画面に遷移) -->
        <isClicked>
                <pushButton>btnReferUpdate</pushButton>
                <event method="referUpdateEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

        <!-- 更新ボタン押下 -->
        <isClicked>
                <pushButton>btnCommitUpdate</pushButton>
                <event method="commitUpdateEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

        <!-- 削除ボタン押下(削除画面に遷移) -->
        <isClicked>
                <pushButton>btnReferDelete</pushButton>
                <event method="referDeleteEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

        <!-- 削除ボタン押下 -->
        <isClicked>
                <pushButton>btnCommitDelete</pushButton>
                <event method="commitDeleteEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

        <!-- 戻るボタン押下 -->
        <isClicked>
                <pushButton>btnReturn</pushButton>
                <event method="returnEvent">todo.event.TodoDetailEvent</event>
        </isClicked>

</action>
```

Event クラスに引き渡すフォームクラスと、そのフォームクラスが、session なのか request なのかを記述する。

処理のコメント

Request の pushButton に設定された値

呼出すイベントクラスのメソッド

Action.xml

Action.xml は画面のボタン押下などの情報を取得し、ボタン押下時の処理の実態を記述する Event クラスの指定メソッドを呼び出す。

【参照ボタン押下時の記述例】

```
<!-- 参照ボタン押下 -->
<isClicked>
        <comment>参照ボタン押下</comment>
        <pushButton>btnRefer</pushButton>
        <event method="referEvent">todo.event.TodoDetailEvent</event>
</isClicked>
```

重　要

アストラル・アパッチの重要なポイントの１つに画面上で発生したイベントでリクエストを伴なうものは、すべて『pushButton』に値が設定されることだ。上記の参照ボタン押下時の例では、『pushButton』の value に btnRefer が設定されていた場合の処理を記述している。

画面から、サーバーへのリクエストが発生するトリガーは常に１つしかない。例えばチェックボックスにチェックしボタン押下しても、それは、チェックボックスという、引数の１つに値が設定されるだけなので、直接のトリガーとはなっていない。

常に HTML の画面を使っている場合では、トリガーは１つであるためアストラル・アパッチではどのトリガーが引かれたのか確認するのは get または post の引数『pushButton』の value を参照する。

HTML 側の処理は以下のとおり。

① PushButton には、押下されたボタンの値が設定される。

```
<input type="hidden" name="pushButton" value="">
```

② 一覧画面から、データを選択し、押下される参照ボタンの HTML 記述

```
<input type="button" name="btnRefer" onClick="jsPushButtonList(this.form, 'btnRefer', '0000000001')" value="参照" >
```

③ 参照ボタン押下時の jsPushButtonList のコード

```
/*　オリジナル（PGM0000004 の派生）
 *　参照ボタンを押下。
*/
function jsPushButtonList(thisForm, buttonValue, id) {

		thisForm.key_todo_code.value = id;
		//alert("ID：" + thisForm.key_todo_code.value);
		this.jsPushButton(thisForm, buttonValue);
		return;
}

/*　PGM0000004
 *　ボタンを押下
 *「hidden」の「pushButton」に検索ボタン名をセットし「submit」する。
 */
function jsPushButton(thisForm, buttonValue) {

		if (buttonValue == 'btnCommitInsert') {
			if (! confirm("登録してよろしいですか？")) { return; }
		}

		if (buttonValue == 'btnCommitUpdate') {
			if (! confirm("保存してよろしいですか？")) { return; }
		}

		if (buttonValue == 'btnCommitDelete') {
			if (! confirm("削除してよろしいですか？")) { return; }
		}

		//自画面を再表示
		thisForm.target = "_self";
		thisForm.pushButton.value = buttonValue;
		thisForm.submit();
		thisForm.focus();
		return;
}
```

重　要

アストラル・アパッチでは、バグのない汎用的な関数は全て PGM1234567 の形式で、型番が採番されている。型番が採番されているソースコード上にバグが存在していた場合、一括でプログラムを修正するなどの、高度な品質管理を実現するためである。よって、型番が採番されているプログラムを修正するときは、コメントから型番を外して修正すること。

<form id="todoDetailForm" scope="session" autoSetScreenParam="true">todo.form.TodoDetailForm</form>の説明。
フォームクラス（form.xml）は各画面に 1 つ存在する。Event クラスに引き渡すフォームクラスと、そのフォームクラスが、session なのか request なのかを記述する。

```
<form id="todoDetailForm" scope="session" autoSetScreenParam="true">todo.form.TodoDetailForm</form>
```

『id』は１つの Action.xml の中で重複しないように記述すれば良いが、システム全体の可読性を考慮すると、システム全体で、Form クラスが同一の場合、『id』は Form クラスが単位で一意性を保っていたほうが良い。つまり、同じ Form クラスであれば、同一『id』でよく、異なる Form クラスであれば、異なる『id』とする。

『scope』は session または request を設定する。session を指定した場合は、Form クラスはセッション上に保持される。request を指定した場合は Form クラスはリクエストとして取り扱われる。

```
<form id="todoDetailForm" scope="session" autoSetScreenParam="false">todo.form.TodoDetailForm</form>
```

『autoSetScreenParam=″true″』はリクエストで上がってきた全ての HTTP パラメータの値を、指定された Form.xml の Data.xml に自働設定する。
リクエストで上がってきた全ての HTTP パラメータの値を、自働設定しない場合は『autoSetScreenParam=″false″』とする。

```
<form id="todoDetailForm" scope="session" autoSetScreenParam="false">todo.form.TodoDetailForm</form>
```

または、『autoSetScreenParam』を記述しなくても、『autoSetScreenParam=″false″』と同様に機能する。

```
<form id="todoDetailForm" scope="session">todo.form.TodoDetailForm</form>
```

<form></form>に記述する、todo.form.TodoDetailForm の部分は、Action.xml が保持する Form クラスのパスを記述する。このパスは、Form クラスのパスであり、Form.xml のパスではないので注意すること。

```
<form id="todoDetailForm" scope="session" autoSetScreenParam="true">todo.form.TodoDetailForm</form>
```

アストラル・アパッチの重要なポイントの１つに画面上で発生したイベントでリクエストを伴なうものは、すべて『pushButton』に値が設定されることだ。上記の参照ボタン押下時の例では、『pushButton』の value に btnRefer が設定されていた場合の処理を記述している。

8. Form.xml の設定内容

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<form>
        <comment>TODO 明細画面</comment>

        <property>
                <comment>TODO 明細</comment>
                <name>todoDetailData</name>
                <data>todo.data.TodoDetailData</data>
        </property>
</form>
```

Form.xml 自体のコメント。

Form.xml の保持する data.xml のコメント。

Data.xml の呼出キー

Data.xml のフルパス

画面上で使用するデータクラスを記述する。１画面に１つ作成する。Form.java に継承させる。
Data.xml は Form.xml に何個でも記述できる。また、同じ Data.xml を複数個記述したい場合は、それぞれの『name』が一意性を保っていれば、別の Data.xml として扱われる。

Form.xml は Form.java に継承させるが、その理由は、Form.java に独自メソッドを記述する必要がある場合に対応するためである。通常は Form.java にメソッドを記述するのは可読性が低くなるので良くない。

9. Form.java の設定内容

JSP へのデータ値受け渡し用のクラス。Form.xml を継承し、１画面に１つ作成する。

```
public class TodoDetailForm extends XMLForm {

        /**
         * デフォルトコンストラクタ
         */
        public TodoDetailForm() {
                super("todo.form.TodoDetailForm");
        }
}
```

Form.xml のパスを記述

Form.xml には、独自のメソッドを追加する事も出来る。可読性が下がるため推奨はできない。

10. Data.xml の設定内容

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<data>

        <comment>TODO 明細データクラス</comment>

        <auto_check_screen_param>

                <pathinfo>TodoDetailData.xml</pathinfo>

<!-- 画面入力項目等 -->
                <property>
                        <name>TODO 番号</name>
                        <http>key_todo_code</http>
                        <html>
                                <show mode="A">
                                        <type>print</type>
                                </show>
                                <show mode="A">
                                        <type>hidden</type>
                                </show>
                        </html>
                        <table>TODO</table>
                        <column>TODO_CODE</column>
                        <comment>TODO を一意に識別する番号</comment>
                </property>

                <property>
                        <name>分類</name>
                        <http>txt_bunrui</http>
                        <html>
                                <show mode="IU">
                                        <type>text</type>
                                        <maxlength>10</maxlength>
                                        <size>25</size>
                                        <ime-mode>active</ime-mode>
                                        <background>#FFFF00;</background>
                                </show>
                                <show mode="SD">
                                        <type>print</type>
                                </show>
                        </html>
                        <validator>
                                <require>TRUE</require>
                                <max_length>20</max_length>
                        </validator>
                        <table>TODO</table>
                        <column>分類</column>
                        <comment>システム名等</comment>
                </property>
        </auto_check_screen_param>
</data>
```

データのキー

Request に設定される name

show mode は、IUDS と IUDS 全てを表す A のいずれかを設定する。また、それ以外のモードの設定もできる。

重　要

<show mode="???">～</show>は画面表示モードを表す(制御する)。アストラル・アパッチの重要概念。アストラルアパッチでは画面表示モードを表す(制御する)するために、<show mode="???">を使用している。通常、I：新規作成（登録)、U：更新、S：参照、D：削除、A：IUSD の全て。その他の特殊な画面モードを作りたい場合は、IUSDA 以外の文字を自由に設定可能。

Data.xml

Data.xml は画面項目名、レングス、属性、各種入力チェック、データベースとのマッピングなどの画面等項目に関する情報を記述する。

```
<property>
        <name>TODO 番号</name>
        <http>key_todo_code</http>
        <html>
                <show mode="A">
                        <type>print</type>
                </show>
                <show mode="A">
                        <type>hidden</type>
                </show>
        </html>
        <table>TODO</table>
        <column>TODO_CODE</column>
        <comment>TODO を一意に識別する番号</comment>
</property>
```

【記述方法の詳細について】
別紙（エクセル）の仕様書を参照すること。

【<show mode="A">が２つ記述されている意味】
TODO 番号の例では、<show mode="A">が２つ記述されて
いるが、常に２つとも出力すると言う意味である。同じ<show mode="A">を複数記述することもできる。

Event.java

```
/**
 * 参照ボタン押下。TODO 明細画面を表示する。
 * @param connection DB コネクション
 * @param statement DB ステートメント
 * @param form Form クラス
 * @return フォワードする JSP へのマップキー(resultmap.xml)
 * @throws Exception
 */
public String referEvent(Connection connection, Statement statement, TodoDetailForm form) throws Exception {

        // 手動で HTTP(POST,GET)の値を取得
        //form.setScreenParam(request, form.getData("todoDetailData"));

        // SELECT todo
        if (! todoDetailBusiness.select(statement, form.getData("todoDetailData"))) {
                return setConfirmForm(request, "選択したデータは削除されています。");
        }

        // 表示モードの設定（参照：S、更新：U、新規作成：I、削除：D)
        form.setShowMode('S');
        return "todo_detail";
}
```

① POST,GET の値取得

② Select 文発行

③ show_mode

④ 戻りの JSP を指定

【処理概要】

Action.xml の『referEvent』が呼びだれたケース。上記例では、TODO 明細画面を表示するために、SQL の Select 文を発行し、データを取得格納する『todoDetailBusiness.select』を呼出している。

① Http の Request 上にある、POST,GET の値取得し、指定した Data.xml に代入する。
Action.xml で『autoSetScreenParam=″true″』にしている場合は、明示的に呼出す必要はなく、XMLAction クラスが呼出された段階で、自働的にセットする。
『autoSetScreenParam=″true″』を指定している場合は、Form.java に記述されている、Data.xml の全てにマッチングする、Request 上の値を設定する。

```
// 手動で HTTP(POST,GET)の値を取得
//form.setScreenParam(request, form.getData("todoDetailData"));
```

【Action.xml の参照ボタン押下時の記述例】

```
<!-- 参照ボタン押下 -->
<isClicked>
        <comment>参照ボタン押下</comment>
        <pushButton>btnRefer</pushButton>
        <event method="referEvent">todo.event.TodoDetailEvent</event>
</isClicked>
```

② SQL の Select 文を発行し、データを取得格納する『todoDetailBusiness.select』を呼出している。データが存在しない場合、『return setConfirmForm(request, "選択したデータは削除されています。");』でエラーメッセージを設定している。

```
// SELECT todo
if (! todoDetailBusiness.select(statement, form.getData("todoDetailData"))) {
        return setConfirmForm(request, "選択したデータは削除されています。");
}
```

③　戻り画面の表示モードを設定している。下記例では『form.setShowMode('S');』となっているので、参照モードで表示することを指定している。

```
// 表示モードの設定（参照：S、更新：U、新規作成：I、削除：D）
form.setShowMode('S');
```

④　戻り画面を指定して処理を終了している。『todo_detail』は JSP の仮想呼出名で resultmap. xml に定義されている。

```
return "todo_detail";
```

Resultmap. xml の『todo_detail』の記述。

```
<!-- TODO 明細画面 -->
<result>todo_detail</result>
<jsppath>/jsp/TodoDetail.jsp</jsppath>
```

Business.java SQL Select 文のケース

```
/**
 * TODO データを 1 レコード取得する。
 * @param statement データベース Statement
 * @param data TODO 明細データクラス
 * @return true：データあり、false：データなし
 * @throws Exception
 */
public boolean select(Statement statement, XMLData data) throws Exception {

	String sqlSlc = ""; String sqlFrm = ""; String sqlWhr = ""; String sqlOrd = "";
	boolean rtn = false;

	/*
	 *  Select todo
	 */
	sqlSlc = "SELECT";
	sqlSlc = sqlSlc + " TODO_CODE";
	sqlSlc = sqlSlc + ", 分類";
	sqlSlc = sqlSlc + ", 内容";
	sqlSlc = sqlSlc + ", 対応方法";
	sqlSlc = sqlSlc + ", 依頼者部課";
	sqlSlc = sqlSlc + ", 依頼者";
	sqlSlc = sqlSlc + ", TO_CHAR(依頼日, 'yyyy/mm/dd') 依頼日";
	sqlSlc = sqlSlc + ", 窓口担当者";
	sqlSlc = sqlSlc + ", TO_CHAR(対応者への依頼日, 'yyyy/mm/dd') 対応者への依頼日";
	sqlSlc = sqlSlc + ", TO_CHAR(期限, 'yyyy/mm/dd') 期限";
	sqlSlc = sqlSlc + ", TO_CHAR(終了日, 'yyyy/mm/dd') 終了日";
	sqlSlc = sqlSlc + ", TODO.依頼内容分類 ID";
	sqlSlc = sqlSlc + ", 依頼内容分類名称";
	sqlSlc = sqlSlc + ", 対応者";
	sqlSlc = sqlSlc + ", 状態";
	sqlSlc = sqlSlc + ", TODO.CREATED";
	sqlSlc = sqlSlc + ", TODO.UPDATED";

	sqlFrm = " FROM TODO";
	sqlFrm = sqlFrm + ", MST_依頼内容分類";

	sqlWhr = " WHERE TODO_CODE = " + os(data.getter("TODO 番号"), CHAR);
	sqlWhr = sqlWhr + " AND TODO.依頼内容分類 ID = MST_依頼内容分類.依頼内容分類 ID (+)";

	printSQL("BusinessClass：TodoDetailBusiness → SQL：" + sqlSlc + sqlFrm + sqlWhr + sqlOrd);
	ResultSet result = statement.executeQuery(sqlSlc + sqlFrm + sqlWhr + sqlOrd);

	if (result.next()) {
		data.setAllColumnValues(result);  // result→データをクラスに自動セット
		rtn = true;
	} else {
		data.initData();
	}
	if (result != null) { result.close(); }
	return rtn;
```

通常の JavaBean である。SQL の Select 文を作成し、発行、『data.setAllColumnValues(result);』でデータ取得結果を Data.xml に代入し正常処理を返す。データが取得できない場合、『data.initData();』で Data.xml を初期化して、エラーを返す。

Business.java　SQL Update 文のケース

```
/**
 * TODO データを 1 レコード更新する。
 * @param statement データベース Statement
 * @param data TODO 明細データクラス
 * @return true： 正常更新、false：予定 UPDATE レコード数と一致しない
 * @throws Exception
 */
public boolean update(Statement statement, XMLData data) throws Exception {

		String sqlUpd = ""; String sqlWhr = "";

		/*
		 *　UPDATE todo
		 */
		sqlUpd = "UPDATE TODO SET ";
		sqlUpd = sqlUpd + "　分類 = " + os(data.getter("分類"), CHAR);
		sqlUpd = sqlUpd + ", 内容 = " + os(data.getter("内容"), CHAR);
		sqlUpd = sqlUpd + ", 対応方法 = " + os(data.getter("対応方法"), CHAR);
		sqlUpd = sqlUpd + ", 依頼者部課 = " + os(data.getter("依頼者部課"), CHAR);
		sqlUpd = sqlUpd + ", 依頼者 = " + os(data.getter("依頼者"), CHAR);
		sqlUpd = sqlUpd + ", 依頼日 = " + os(data.getter("依頼日"), DATE8);
		sqlUpd = sqlUpd + ", 窓口担当者 = " + os(data.getter("窓口担当者"), CHAR);
		sqlUpd = sqlUpd + ", 対応者への依頼日 = " + os(data.getter("対応者への依頼日"), CHAR);
		sqlUpd = sqlUpd + ", 期限 = " + os(data.getter("期限"), DATE8);
		sqlUpd = sqlUpd + ", 終了日 = " + os(data.getter("終了日"), DATE8);
		sqlUpd = sqlUpd + ", 依頼内容分類 ID = " + os(data.getter("依頼内容分類 ID"), CHAR);
		sqlUpd = sqlUpd + ", 対応者 = " + os(data.getter("対応者"), CHAR);
		sqlUpd = sqlUpd + ", 状態 = " + os(data.getter("状態"), CHAR);
		sqlUpd = sqlUpd + ", UPDATED = SYSDATE";

		sqlWhr = " WHERE TODO_CODE = " + os(data.getter("TODO 番号"), CHAR);

		printSQL("BusinessClass：TodoDetailBusiness → SQL：" + sqlUpd + sqlWhr);
		int rtn = statement.executeUpdate(sqlUpd + sqlWhr);

		if (rtn == 1) {
			return true;
		}
		return false;　//予定 UPDATE レコード数と一致しない
}
```

通常の JavaBean である。SQL の Update 文を作成し、発行、Update して件数が戻り値で返るので、Update 文の戻り値と、想定される Update レコード件数を比較し、『==』のケースで通常処理終了。一致しない場合はエラーで処理を終了している。

重　要

アストラル・アパッチのフレームワーク部分は append などを適切に使い分けているが、アプリケーションごとに開発するクラスに関しては append を推奨していない。理由は可読性と他言語への移行が複雑になるからである。『+』　で連結をしていれば、読みやすく、また、高速化のために C++などへの移行も速やかに行える。重要なことは、可読性が高く、メンテナンスが容易であることで、著しくパフォーマンスを落とさないのであれば、言語特有の表記は避けるべきである。

この点で異論があるケースは自由に記入すればよいと思うが、マネジメント層の人間とよく協議をし、プログラマ目線ではなくマネジメント目線でシステムをとらえ最適化を図る事が望ましい。