授業評価システムにおけるオブジェクト指向の設計について

蒋　　湧 *・湯川治敏 *・龍　昌治 **

* 愛知大学経済学部　** 愛知大学短期大学部

はじめに

大学における FD 活動の一環として、学生により授業評価はアンケートの形で行われていた。今まで紙ベースで実施した授業アンケートをより効率的、正確的、経済的に行うために、Web ベースの授業評価システムを開発した。このシステムを Web Based Class Evaluation System と名づけ、本文には WBCES と簡略する。

WBCES システムはデータベースサーバー、ウェブサーバーとクライアント（携帯電話を含め）により構成した典型的な 3 層ウェブアプリケーションである（図 1)。サーバーは Linux 環境で作動し、

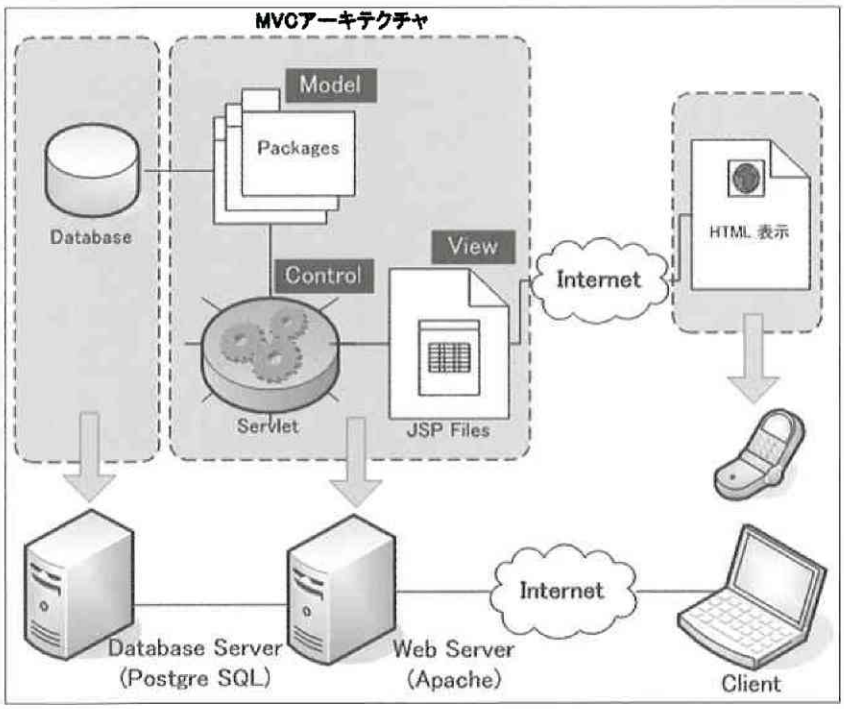

図 1　WBCES のシステム構成

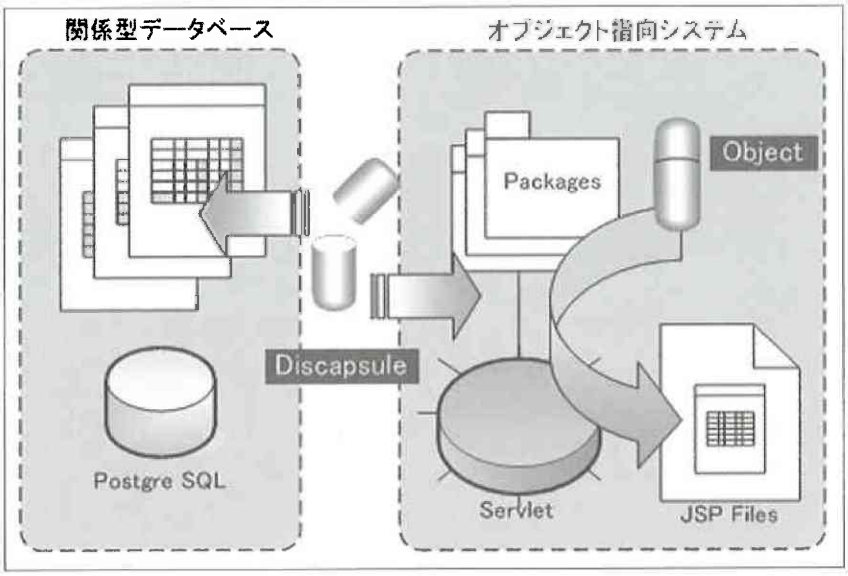

図2　オブジェクトとデータベースの間のデータマッピング作業

ウェブサーバーとしてApacheを、データベースサーバーとしてPostgreSQLを採用した。システムアーキテクチャにMVC（Model、View、Control）モデルが導入され、JSP（Java Server Pages）によりHTML生成処理のロジックとJavaBeansによりアンケート処理のロジックが分離されたことで、より優れたシステムの保守性とコードの再利用性を目指した。

開発環境については、Eclipse3を中心に、Tomcat、EclipseUML、PhpAdmin、PgAdminなどOpen Sourceが採択され、高品質、高機能なオブジェクト指向プログラミング環境とデータベース構築・操作環境が無償で実現された。

WBCESは関係型データベースを利用したオブジェクト指向のウェブアプリケーション・システムである。WBCESにもっとも頻繁的に行われるのは、オブジェクトとデータベースの間に行われるデータマッピング作業である。たとえば、ユーザー認証時、データベースから抽出したユーザー情報をユーザーオブジェクトに実装する作業、或いは、回答オブジェクトに含まれている回答データを取り出し、関連テーブルに書き込む作業など、様々なオブジェクトがデータベースとアクセスしながら、データの読み取り、書き込みを行う（図2）。これらの作業はウェブアプリケーションにとって、もっとも基本的な作業である。ときには大量なデータベースアクセス作業が集中的に行われるため、システムの負荷と信頼性に影響を与えかねない。

本文では、多くのオブジェクトに共有できるデータベースプログラミングのデザインパターンを提案する。WBCESシステムでの実験結果により、データベースプログラムの再利用性が極めて向上し、より優れたシステム信頼性と保守性が得られた。また、このデータベースプログラミングのデザインパターンは汎用性を持つ、広範囲のJavaウェブアプリケーションに利用できる。

本文は次のように構成される。第1節には授業アンケートシステムの概要を述べる。データ構造を含めたデータベースモデルは第2節に説明する。第3節には標準的なデータベースプログラミングを

紹介し、第4節には共有できるデータベースプログラミングデザインパターンを提案する。最後の第5節には、提案したデータベースプログラムパターンの応用事例を紹介する。

1. 授業アンケートシステムの概要

授業評価システムには、学生、教員と管理員、3種類のユーザーがあり、そのうち管理員は数名の教員が兼任することを想定している。全てのユーザーは共通のログイン画面を利用し、ユーザー認証を行う。学生は携帯電話によるアクセスも可能である。認証に成功した場合、それぞれ学生と教員用の画面へ移り、ユーザー権限によりアンケートの実施やアンケートに関わる初期設定などができる(図3)。管理権限を持つ教員の画面には、管理員画面へのログインリンクが自動的に表示される。管理員としての認証が成功した場合、管理員の画面に移る。図3には、学生、教員と管理員、ユーザーごとのシステムと主な機能を表示している。

学生画面には、学生個人の履修科目が学期ごとにリストされている。各々の科目名と伴に、該当科目のアンケート実施状態も明示されている。例えば、アンケートの回答期間において、まだ回答していない科目に対し、アンケート状態は「未回答」と明記すると同時に、アンケート回答画面へのリンクも自動的に張られる。科目名をクリックするだけで、アンケートを回答することができる。回答完了後、リンクは自動的に消え、アンケート状態は「回答済み」と変わる。

教員画面には、教員個人の担当科目が学期ごとにリストされている。アンケートの進行状態により画面上適切な内容を表示しながら、対応した機能を提供している。たとえば、アンケート実施日の変

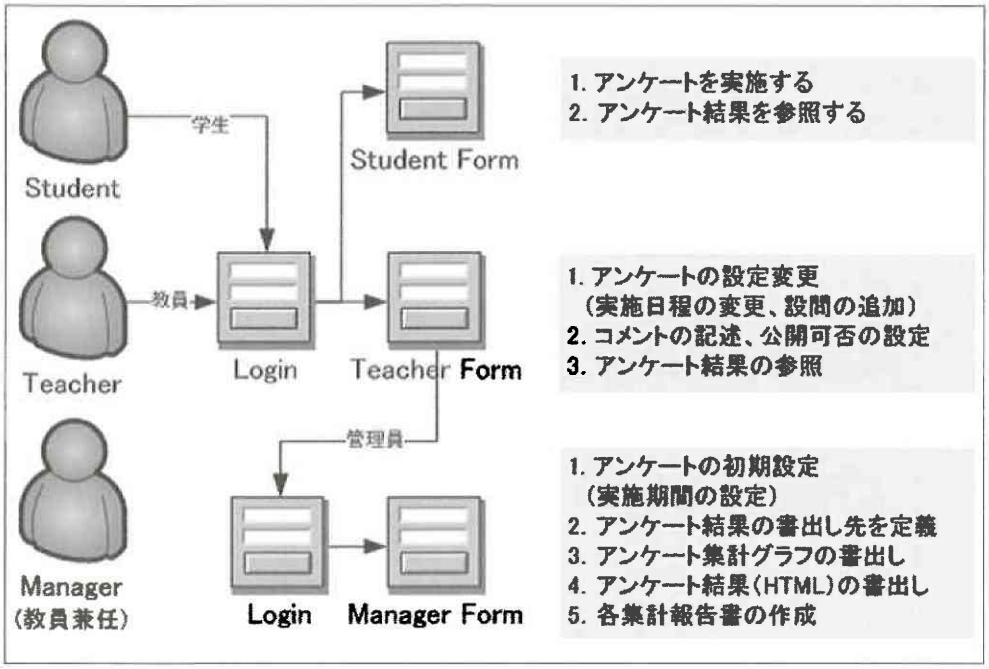

図3　ユーザータイプとシステムの主な機能

更作業と設問の追加作業は、回答期間内には行えない。また、回答期間後、科目ごとのアンケート結果と科目群の平均結果が集計され、図表形式で確認することができる。その結果に基づいて、教員によりコメントを記述し、そのアンケート結果を公表するかどうかについて、教員自ら選択することが可能である。

学生と教員の画面には、今までのアンケート結果を参照するボタンが配置され、本人の履修科目だけではなく、全ての開講科目のアンケート結果を参照することができる。

管理員サイトでは、アンケートの初期設定、集計グラフの書出し、アンケート集計と結果公表に関わる作業が行われる。初期設定には、アンケート実施期間の設定と回答データの保存先の設定が含まれている。設定した項目はデータベースに保存され、アンケート実施の基準データになる。

アンケートの結果報告書はウェブページで表示される。科目ごとの回答結果と科目群の平均値結果が集計され、図表の形で公表する。大量な集計作業を要するため、1ページを表示するまで多くの時間がかかる。ページ表示のレスポンスタイムを最小限に減らすために、集計グラフを予めJPGの形式で一括書き出すツールを管理員サイトに用意した。また、管理員サイトには、各種のアンケート報告書を作成するために、多くの集計関数を用意した。従来の紙ベースのアンケートと比べ、より正確、より多種類の集計データを効率的に得られるようになった。

2. データベースモデル

授業評価のワークフローを踏まえて、関わるデータとデータの性質、さらにデータ間の関連を整理しながら、データベースのデータ構造を探っていく。授業評価に参加するのは学生と教員であり、教員が開講した全ての講義は評価の対象となる。下図のように、授業評価に関わる全てのデータは、「学生」、「教員」、「科目」、この3つの実体データの元から派生される。

まず、科目と担当教員さえあれば、講義を開講することができる。その場合、開講科目名、担当教員、開講学期、開講日と開講形態（集中講義など）は開講データとして生じる。次に開講した講義に対し、学生の履修が現れる。履修データとして、誰がどの講義を履修するかについて記述される。ま

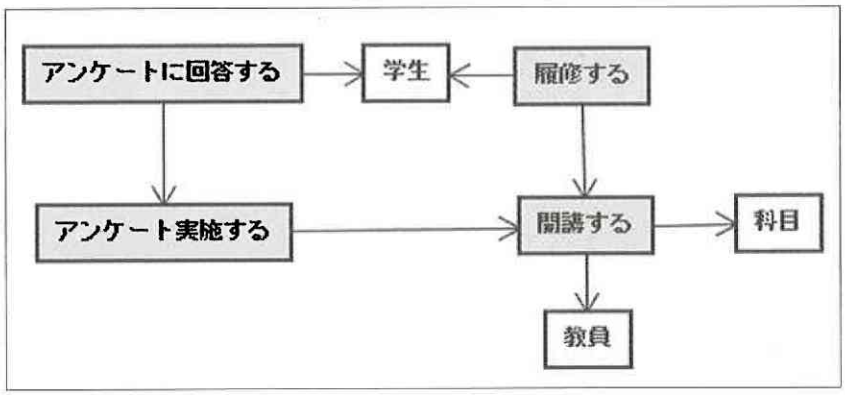

図4　授業評価に関わるデータとデータの性質

た、開講した全ての講義は授業評価の対象となり、学生を対象に授業アンケートが実施される。最後に、実施しているアンケートに学生が回答するので、回答データが集まる。このワークフローに現れる「開講する」、「履修する」、「アンケートを実施する」と「アンケートに回答する」は、それぞれ開講、履修、アンケート実施とアンケート回答に関わる4つのイベントが発生するたびに、生じる事象データである。

ワークフローに基づく、詳細な論理分析やクラス構造全貌の紹介は、紙幅上の理由で省略する。クラス構造に永続性のあるエンティティを抽出し、構築したデータベースモデルは下図のように表わせる。

学生、教員のデータはそれぞれ students、teachers テーブルに格納する。学生の識別子は学籍番号（studentid）、教員の識別子は教員番号（teacherid）に設定し、学生と教員を一意的に識別することができる。学生は学籍番号（studentid）で、教員は loginname でログインする。

科目のデータは subjects テーブルに保存している。授業アンケートは科目群ごとにタイプを定めているので、科目のアンケート区分は外部キーの inquirytypeid を通して、inquirytype テーブルと関連をつける。アンケートタイプ名は科目群名を利用している。アンケートの設問内容と設問数はアンケートタイプごとに異なっている。そのため、配列型の que_length フィールドの中には、設問数が格納されている。

開講科目のデータは classes テーブルに格納している。classid と termno（学期番号）を識別子とし

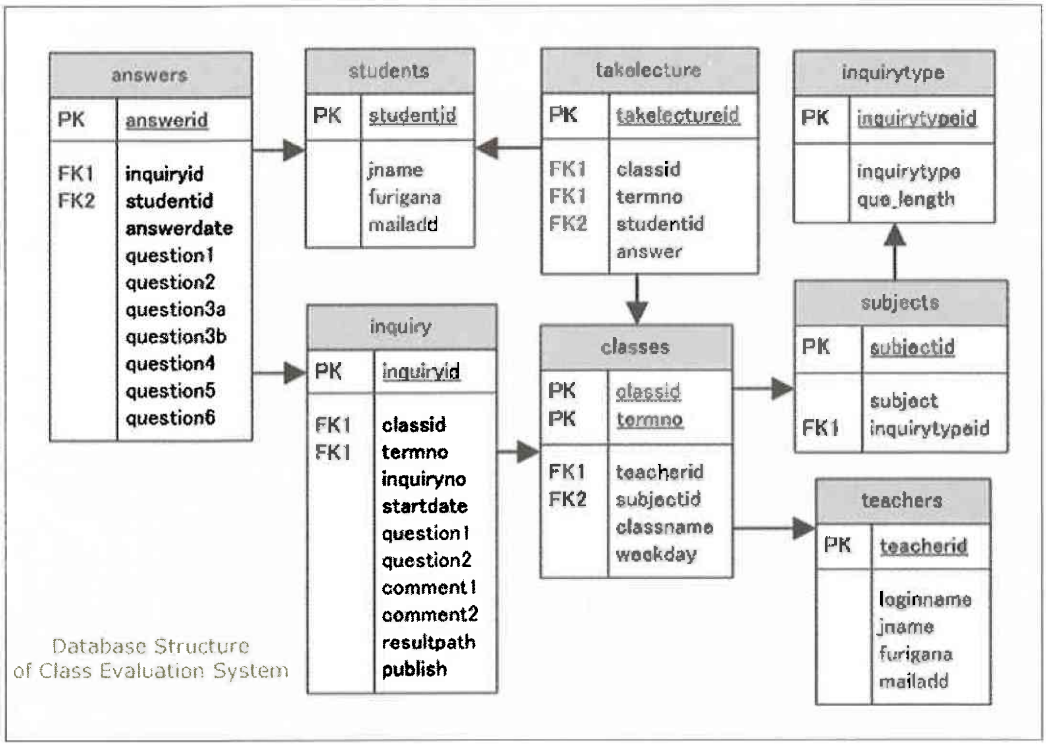

図5　WBCES のデータベースモデル

て指定するため、同一の科目の通年開講問題は解決できる。開講テーブルには、担当教員の情報、科目の情報、開講日の情報が欠かせないため、外部キーteacheridとsubjectidを通して、それぞれ教員と科目テーブルに関連をつけている。

履修データはtakelectureテーブルに記述し、どの学生がどの講義を履修するかを定めるために、studentidとclassid、termnoでそれぞれ学生テーブルと開講テーブルに関連づける。アンケートへ回答記録はanswerフィールドに記述する。

アンケートの実施データはinquiryテーブルに格納する。開講テーブルと関連づけるためにclassidとtermno外部キーが設定している。中期、後期アンケートの識別はinquirynoで判断し、アンケート結果の公表可否はpublishで決める。公表する場合、公表データの保管パスはresultpathに記憶する。アンケートの開始日、追加設問と教員コメントはそれぞれstartdate、question1～question2とcomment1～comment2フィールドで収める。

最後に、アンケートの回答データはanswersテーブルに保存される。だれが、いつ、どのアンケートに回答したかについて、それぞれstudentid、answerdateとinquiryidフィールドに記述する。各問題への回答は配列型のフィールドquestion1～question6に格納する。

図5で示したデータベースモデル以外に、WBCESには15のデータビューを設け、いろいろなケースのフォームプレゼンテーションに備える。そのうちの一つとして、開講科目・アンケート情報を総合したvinquiryclassデータビューを紹介する。下図に示されたように、このデータビューはinquiryテーブルをベースにし、更にclasses、subject、inquirytypeとteachers、4つのテーブルの情報を加えて作ったデータビューである。

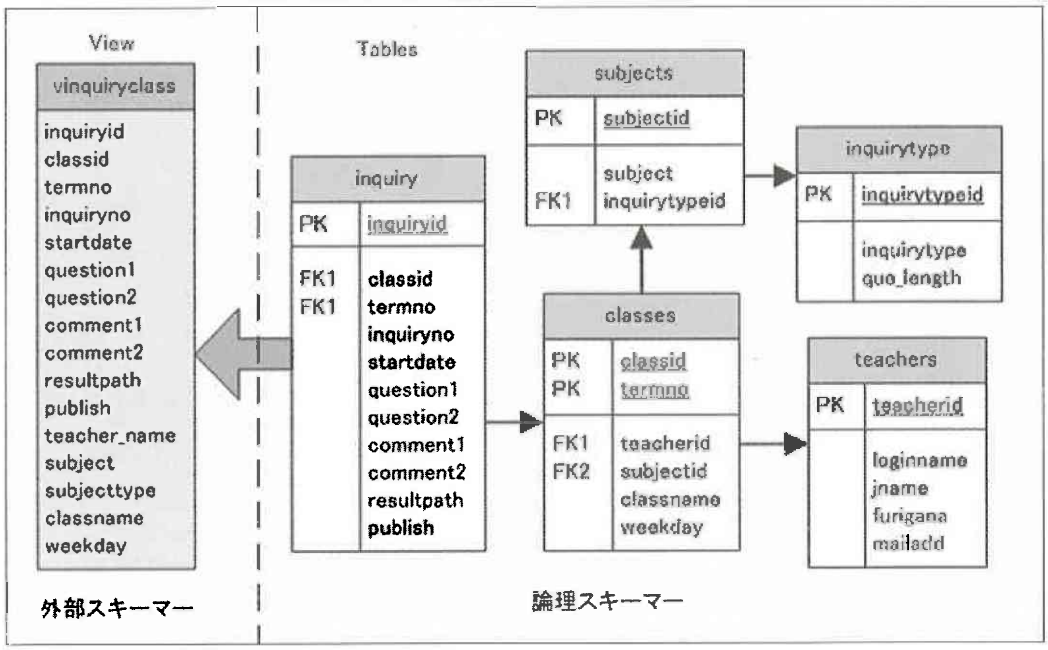

図6　データビューvinquiryclassの構成イメージ

データの集計はデータベースシステムのもう一つ重要な機能である。WBCESには、大量な集計作業に備えるために、47のデータベース関数をPostgreSQL Store Procedureで定義し、データ集計の効率と能力を向上させた。

3. 標準的なJavaデータベースプログラミングコード

近年、WECESのようなデータベースと連携したウェブアプリケーションが非常に増えているため、Javaデータベースプログラミングも標準的になりつつある。SQL（Structured Query Language）によるテーブルの基本操作とJDBC（Java Database Connectivity）によるデータベースへのアクセスは、Javaデータベースプログラミングの基本と言える。SQL文を利用したテーブルの操作はSelect、Insert、UpdateとDelete4種類の基本クエリが含まれている。それらのクエリはJDBCを通して、データベースへ送り、その結果は再びJDBCを経由し、受けとるという流れになる。

図7では「教員テーブルから専任教員全員のデータを取り出す」という事例を取り上げ、標準的なJavaデータベースプログラミングコードを検証してみる。

1行にはjava.sqlパッケージで規定されているクラスをImportすることで、JDBCドライブに実装される機能を使える環境に整える。

```
1.  import java.sql.*;

2.  private final String
    url="jdbc:postgresql://localhost:5432/wbcesdb?useUnicode=true&characterEncoding=EUC-JP";
3.  private final String user="aichi_user";
4.  private final String password="aichi_pwd";

5.  Class.forName("org.postgresql.Driver");
6.  Connection cn = DriverManager.getConnection(this.url,this.user,this.password);
7.  Statement st = cn.createStatement();
8.  String sql =" Select * from teachers where teacherid like '1100%'";
9.  ResultSet rs = st.executeQuery(sql);
10. ArrayList aTeacherList = new ArrayList();
11. String[] aTeacher;
12. aTeacher = new String[4];
13.    while(rs.next()){
14.              aTeacher[0]=rs.getString("teacherid");
15.              aTeacher[1]=rs.getString("loginname");
16.              aTeacher[2]=rs.getString("jname");
17.              aTeacher[3]=rs.getString("furigana");
18.              aTeacher[4]=rs.getString("mailadd");
19.              aTeacherList.add(aTeacher);
20.    }
21. rs.close();
22. st.close();
23. cn.close();
```

図7　標準的なJavaデータベースプログラミングコード(1)：Selectケース

2行～6行では、JDBCを利用しデータベースへの接続を行う。そのうち、5行目にPostgreSQL用のJDBCドライバをロードし、続いて第6行目では、localhostというデータベースサーバーにあるwbcesdbというデータベースに接続を行う。そのとき、データベースユーザー、パスワードと文字コードを設定する必要がある。この接続オブジェクトはcnと定義している。

第7～9行ではJDBCを通してSQLクエリをデータベースへ送信し、データベースからの検索結果を受け取る。まず、第7行目では、SQLを送信するために使われるConnectionクラスのStatementオブジェクトをstと定義し、次にSQLクエリを文字列sqlに記述する。最後に第9行目で、st.executeQuery(sql)でSQLクエリが送信され、その結果をResultSetオブジェクトのrsで受け取る。

10行～20行では、ResultSetオブジェクトrsから受け取った教員リストを配列aTeacherListに格納する。

21行～23行には、ResultSet、StatementとConnectionの順に、それぞれのオブジェクトを解放し、データベースとの接続を切断する。

次に、以下のデータを更新するためのJavaデータベースプログラミングコードを検証する。

図8に示したコードと図7で紹介した検索コードを比較してみると、データベース接続部分のコードはまったく同じことがわかる。異なるのは以下の3箇所だけである。

1. 第8行で記述したSQLクエリの内容が違う。
2. 第9行に、SQLを実行するためにst.executeUpdate(sql)オブジェクトが使われる。図7の場合、st.executeQuery(sql)を使用し、Selectタイプの SQLクエリを実行する。
3. 更新（Update）クエリには、通常返される結果はないため、ResultSetオブジェクトは使用しない。

追加（Insert）と削除（Delete）タイプのJavaデータベースプログラミングコードも更新（Update）タイプのコードとほとんど同様であり、修正に必要な箇所は、図8の第8行のSQL文の内容だけであ

```
1.  import java.sql.*;

2.  private final String
    url="jdbc:postgresql://localhost:5432/wbcesdb?useUnicode=true&characterEncoding=EUC-JP";
3.  private final String user="aichi_user";
4.  private final String password="aichi_pwd";

5.  Class.forName("org.postgresql.Driver");
6.  Connection cn = DriverManager.getConnection(this.url,this.user,this.password);
7.  Statement st = cn.createStatement();
8.  String sql="UPDATE teachers SET loginname='taro', jname=' 愛 知 太 郎 ', furinaga='Taro Aichi',
    mailadd='taro@aichi-u.ac.jp' WHERE teacherid ='1100111' ";
9.  st.executeUpdate(sql);
10. st.close();
11. cn.close();
```

図8　標準的なJavaデータベースプログラミングコード(2)：Updateケース

る。

上述のコード分析で、標準Javaデータベースプログラミングに何らかの構文パターンが存在してることが確認された。JDBCパッケージの使用は、固定したプログラムパターンが現われる主な原因である。しかし、SQLクエリは各々の事例により異なっているので、SQL文の実行に使われるオブジェクトは異なる。次の節には、このプログラムパターンを活かし、オブジェクト指向のJavaデータベースプログラミングのデザインパターンを提案する。

4. オブジェクト指向のJavaデータベースプログラミングのデザインパターン

4.1 WBCESにおけるオブジェクト指向デザインの概要

授業評価システムWBCESのアンケートロジック部分の設計は、オブジェクト指向の手法を取り入れた。システム全体は、下図に表れるように8つのパッケージで構成される。そのうち、Teacher、Student、Manager、Inquiry、Subject_ClassとReportは、何らかの実体と対応するクラスで構成したパッケージであり、一方DAO（Database Access Objects）パッケージとUtilityパッケージは補助機能を提供するクラスにより構成され、データベースアクセス機能やJSPファイルの間に情報伝達などの機能を提供している。

表1には各にパッケージに含まれる主なクラスを示す。

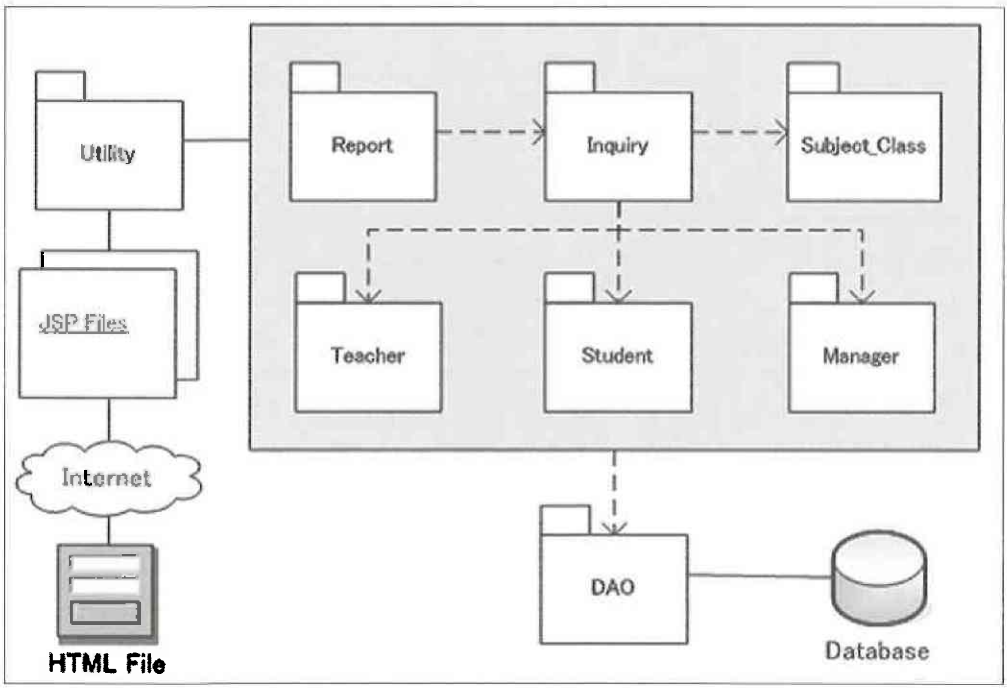

図9　WBCESシステムのパッケージ構成

表1　WBCES システムの主なクラス構成

パッケージ名.クラス名	説　明
Teacher.cTeacher	単独教員のクラス（Teachers テーブルに対応）
Teacher.beanTeacherList	教員リストのクラス
Student.cStudent	単独学生のクラス（Students テーブルに対応）
Student.beanStudentList	学生リストのクラス
Manager.cManager	単独管理員のクラス
Subject_Class.cSubject	単独科目のクラス（Subjects テーブルに対応）
Subject_Class.beanSubjectList	科目リストのクラス
Subject_Class.cTakeLecture	履修科目クラス（TakeLecture テーブルに対応）
Subject_Class.cClasses	開講科目のクラス（Classes テーブルに対応）
Subject_Class.beanClassesList	開講科目リストクラス
Subject_Class.ClassStudent	履修科目・アンケートの回答状態を扱うクラス （vTakeLecture データビューに対応）
Subject_Class.ClassTeacher	担当科目・アンケート実施情報を扱うクラス（vClass データビューに対応）
Inquiry.cInquiry	単独アンケートのクラス（Inquiry テーブルに対応）
Inquiry.beanInquiryList	アンケートリストクラス
Inquiry.cInquiryType	単独アンケートタイプのクラス（InquiryType テーブルに対応）
Inquiry.cAnswers	単独回答のクラス（Answers テーブルに対応）
Inquiry.beanAnswerList	回答アンケートリストクラス
Inquiry.cInquiryClass	開講講義・アンケート実施情報を扱うクラス(vInquiry データビューに対応)
Inquiry.cStatisticalReport	アンケート回答を集計するクラス(vStatistical_Reports データビューに対応)
Report.cInquiryPublish	公表するアンケートを集計するクラス（Store Procedure に対応）
Report.cInquiryAnswer	学生におけるアンケート回答の状況を集計するクラス(Store Procedure に対応)
Report.cInquiryTeacher	教員におけるアンケート実施の状況を集計するクラス(Store Procedure に対応)
DAO.cDBConnection	データベース接続機能を提供するクラス
DAO.cDAO	SQL クエリと Store Procedure を実施するクラス
DAO.cGetObjectFromDB	データベースから Object リストを引出すクラス
Utility.svltWebCES	サーブレットクラス
Utility.JaasKrbBean	ユーザー認証クラス（Open Source）
Utility.cFile	フォールダ、ファイルを扱うクラス
Utility.ctagDataTable	カスタマー・タグを作成するクラス
Utility.cHTMLWriter	HTML ファイルの書き出すクラス
Utility.cDrawAnswerChart	集計グラフの書く出すクラス
Utility.JFreeChartBean	グラフの基本要素クラス（Open Source）

4.2 WBCESにおけるオブジェクト指向の設計思想

WEBCESにおけるクラスの設計は、次のような考えに沿って開発してきた。

I. 基本クラス（Basic Class）の定義

データベースモデルを踏まえて、永続性のある実体データと実体データを対応したオブジェクトを中心に、基本クラス群を定義する。それらのクラスは、単独のオブジェクトの集合であり、その属性は対応するデータベース・テーブルのフィールドと一致する。テーブルとのやり取りに備え、Insert、Delete、UpdateとSelectなどSQL文を実行するメソッドを実装する。上述の表1の中に、Teacher.cTeacher、Student.cStudent、Subject_Class.cClassesとInquiry.cInquiryなどは基本クラスに属する。

II. 基本クラスからリストクラス（List Class）へ拡張

前述の基本クラス群は単独のオブジェクトに注目したクラス群である。たとえば、一名の教員、一人の学生、一科目と一回アンケートなどがあげられる。一方、WBCESには同類のオブジェクトの集合も頻繁に使われる。たとえば、ある教員の担当科目一覧、ある学生の履修科目一覧、予定しているアンケートのリストとアンケート回答リストなどは具体化したオブジェクトの集合になる。このようなオブジェクトの集合も効率よく作り出すために、各種のリストクラスを定義した。リストクラスは、前述の基本クラスを使用し、ArrayListで基本クラスの配列を返すことになる。その意味で、リストクラスは基本クラスの拡張と言える。たとえば、基本クラスのTeacher.cTeacherに対し、その拡張Teacher.beanTeacherListはリストクラスになる。

III. 基本クラスからプレゼンテーションクラス（Presentation Class）へ拡張

次にデータベースのデータビューに注目し、データビューごとのプレゼンテーションクラスを定義する。複数のテーブルから必要な情報を抽出し、形成したのはデータビューである。ほとんどのユーザーインターフェースで表示したのはデータビューの情報である。MVC型のJavaウェブアプリケーションの場合、データビューの情報こそ、サーブレットを経由し、JSPまで伝達する必要がある。オブジェクト指向の観点から見れば、プレゼンテーションクラスも基本クラスを特化したクラスである。たとえば、プレゼンテーションクラスSubject_Class.ClassStudentは、基本クラスSubject_Class.cTakeLectureを継承し、更にInquiryの情報を加え、拡張したクラスである。このクラスは学生画面に学期ごとの履修科目一覧と対応したアンケート実施状態を表示するために使われる。

IV. 集計クラスの定義

上述のⅠからⅢのクラスは、基本クラスのFamilyであり、以下の2つの共通点を持つ。

① クラスの属性は、何らかの実体データ、或いは事象データと対応する。

② クラスのメソッドには、SQL文を実行するため機能を実装している。

WBCESの集計クラスは、さまざまなアンケート結果を集計するために定義したクラスであるので、基本クラスを継承する必要はない。なお、データの集計を効率的に行われるために、直接データベース内部のStore Procedureを駆動した集計メソッドを実装した。表1のReport.cIn-

quiryPublish、Report.cInquiryAnswer と Report.cInquiryTeacher は集計クラスである。

V. 補助クラスの追加

既に述べた DAO パッケージと Utikity パッケージに含まれたクラスは、補助クラスである。特に、Utility.svltWebCES クラスは、サーブレッドクラスとして、MVC モデルの中にコントロールの役割を果たしている。また DAO パッケージに含まれる 2 つのクラスは抽象クラスとして定義されている。それぞれ備えている SQL 実行とオブジェクトのリスト化機能を、全ての基本クラスに継承させるため、コードの再利用性は極めて高いレベルに達した。この部分の詳細について、次の節に解説する。

4.3 WBCES におけるオブジェクト指向のデータベースプログラムデザインパターン

全ての基本クラスは SQL 文を実行するメソッドが実装されている。データベースのテーブルとのやり取りの際に、1 オブジェクトがテーブルの 1 レコード（1 行）とデータマッピングしながら、情報の取り出しや書き込みが行われる。性格の違うオブジェクトに対して、当然データマッピングコードも異なってくる。ただし、共通の JDBC を利用しているため、第 3 節で検討したプログラムのパターンが、どこまで最大限に共有できるかは、設計の最大目標であった。

全ての基本クラスに対し、データベースアクセス、SQL 文の実行とオブジェクトのリスト化、3 つの機能を提供するクラス群を次の DAO パッケージとしてまとめた。

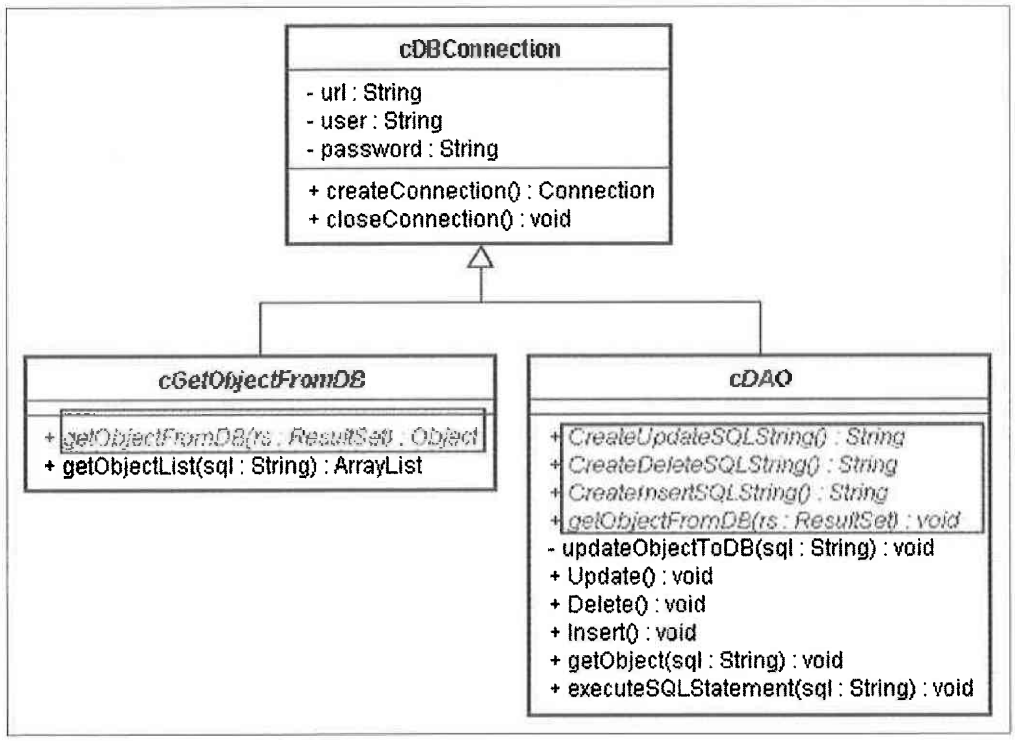

図 10 DAO パッケージ構成

DAOパッケージは3つのクラスにより構成される。cDBConnectionはデータベースへのアクセス、cDAOはSQL文の実行、cGetObjectFromDBはオブジェクトリストの作成、3つの役割を分担している。データベースへのアクセスは、全ての作業に関わっているので、cDBConnectionクラスをcDAOクラスとcGetObjectFromDBクラスに継承させる。

cDBConnectionクラスには、Connection型のcreateConnection()関数とcloseConnectionメソッドを実装している。データベースに接続するためのパラメーターを一箇所に設定することで、テスト・管理上のメリットは大きい。

クラスcDAOは、cDBConnectionを継承した抽象クラスである（図12)。外部に対しUpdate（行22～行24)、Delete（行25～行27)、Insert（行28～行30)、getObject（行31～行48）とexecuteSQLstatement（行49～行59)、5つのメソッドを提供する。前の4つのメソッドは、Update、Delete、InsertとSelect、4つのタイプのSQLクエリに対する実行機能を提供し、5つ目のexecuteSQLstatementはデータベースStore Procedureを駆動するためのメソッドである。あらゆるオブジェクトに対応できるためには、SQLクエリの記述部分（行4～行6）とデータマッピング部分

```
1.  package aichi.wbces.dao;
2.  import java.sql.*;

3.  public class cDBConnection {
4.  private final String

    url="jdbc:postgresql://localhost:5432/wbcesdb?useUnicode=true&characterEncoding=EUC-JP";
5.   private final String user="aichi_user";
6.   private final String password="aichi_pwd";

7.   public cDBConnection(){};

8.   public Connection createConnection(){
9.     try {
10.          Class.forName("org.postgresql.Driver");
11.          Connection cn = DriverManager.getConnection(this.url,this.user,this.password);
12.          return cn;
13.       } catch (ClassNotFoundException e) {
14.          System.out.println("Can not found JDBC Driver. ¥n");
15.       } catch (SQLException e) {
16.          System.out.println("Connect Error. ¥n");
17.       }
18.       return null;
19.  }

20. public void closeConnection(Connection cn) {
21.   try {
22.        cn.close();
23.    } catch (Exception ex) {}
24.    }
25. }
```

図11　cDBConnectionクラスのコード

（行 7）は抽象的に宣言し、コードの実装はそれぞれの基本クラスに譲り渡す。

```
1. package aichi.wbces.dao;
2. import java.sql.*;

3. public abstract class cDAO extends cDBConnection{
4.    public abstract String CreateUpdateSQLString();
5.    public abstract String CreateDeleteSQLString();
6.    public abstract String CreateInsertSQLString();
7.    public abstract void getObjectFromDB (ResultSet rs);

8. public cDAO(){}

9. private void updateObjectToDB(String sql){
10.       Connection cn = null;
11.       try {
12. cn=this.createConnection();
13.            Statement st = cn.createStatement();
14.            st.executeUpdate(sql);
15.       }
16.       catch (Exception e) {
17. e.printStackTrace();
18.       } finally {
19. this.closeConnection(cn);
20.       }
21.    }

22. public void Update(){
23.       this.updateObjectToDB(this.CreateUpdateSQLString());
24. };

25. public void Delete(){
26.       this.updateObjectToDB(this.CreateDeleteSQLString());
27. };

28. public void Insert(){
29.       this.updateObjectToDB(this.CreateInsertSQLString());
30. };

31. public void getObject(String sql){
32.    Connection cn = null;
33.    try {
34.          cn = this.createConnection();
35.          Statement st = cn.createStatement();
36.          ResultSet rs = st.executeQuery(sql);
37.          while(rs.next()){
38.             this.getObjectFromDB(rs);
39.          }
40.          rs.close();
41.          st.close();
42.    }
43.    catch (Exception e) {
44.          e.printStackTrace();
```

```
45.    } finally {
46.        this.closeConnection(cn);
47.    }
48. }

49. public void executeSQLstatement(String sql){
50.    Connection cn=this.createConnection();
51.    try {
52.        Statement st = cn.createStatement();
53.        st.execute(sql);
54.    }
55.    catch (Exception e) {
56.        e.printStackTrace();
57.    } finally {
58.        this.closeConnection(cn);
59.    }
60. }
}
```

図 12　cDAO クラスのコード

```
1. package aichi.wbces.dao;
2. import java.sql.*;
3. import java.util.*;

4. public abstract class cGetObjectFromDB extends cDBConnection {

5. public cGetObjectFromDB(){};
6. public abstract Object getObjectFromDB (ResultSet rs);

7. public ArrayList getObjectList(String sql){
8.    ArrayList ResultList= new ArrayList();
9.    Connection cn = null;
10.   try {
11.       cn = this.createConnection();
12.       Statement st = cn.createStatement();
13.       ResultSet rs = st.executeQuery(sql);
14.       while (rs.next()) {
15.           ResultList.add(this.getObjectFromDB(rs));
16.       }
17.     rs.close();
18.     st.close();
19.     return ResultList;
20.   }
21.   catch (Exception e) {
22.   e.printStackTrace();
23.   return null;
24.   } finally {
25.       this.closeConnection(cn);
26.   }
27.  }
}
```

図 13　cGetObjectFromDB クラスのコード

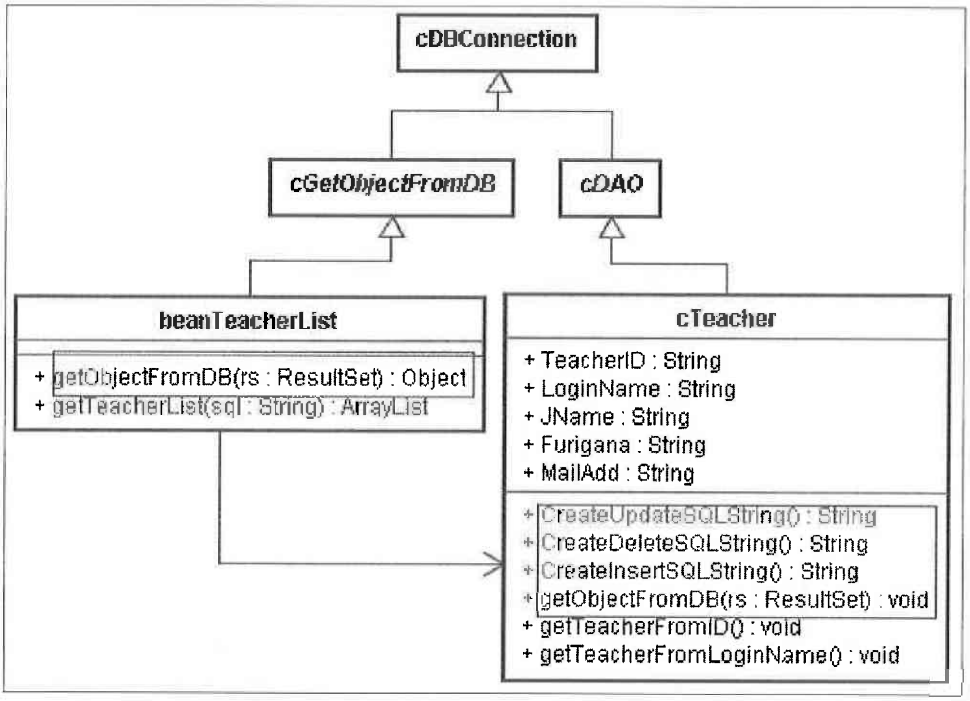

図 14　DAO を継承した教員クラスと教員リストクラス

クラス cGetObjectFromDB は、cDAO と同様に cDBConnection を継承した抽象クラスである。複数のオブジェクトを抽出し、配列型のオブジェクトリストを返すことは getObjectList（行 7～行 27）メソッドの任務である。そのうち、各々のオブジェクトは第 15 行の this.getObjectFromDB で受け取っているが、このメソッドも抽象メソッドとして第 6 行で定義され、その中のデータマッピングの実装は各基本クラスに譲る。

以上の DAO パッケージを定義した後、次には基本クラス（プレゼンテーションクラスも同様）とリストクラスがどのように DAO を継承するかについて、教員クラスの例で解説する。

基本クラス cTeacher は cDAO 抽象クラスを継承しているので、cDAO で定義された抽象メソッドを実装する必要がある。また cDAO に持っている Update、Delete、Insert などのメソッドは使える。一方、リストクラス beanTeacherList クラスは cGetObjectFromDB を継承しながら、cTeacher のオブジェクトを次々と ArrayList に取り入れ、cTeacher のリストを配列型で返される。

図 14 に示されたように、JDBC に関わるほとんどの作業が DAO パッケージに集約され、教員クラスに特化した作業は cTeacher と beanTeacherList に残されている。そのため、DAO パッケージと Teacher パッケージは完全に分離することができ、お互い独立していることが確認できる。その DAO パッケージの独立性によって、優れたシステムの保守性とコードの再利用性を得られた。

```
1.  package aichi.wbces.teacher;
2.  import aichi.wbces.dao.*;
3.  import java.sql.*;
4.  import java.util.Date;
5.  import java.util.*;
6.  import java.text.*;

7.  public class cTeacher extends cDAO {

8.  private String TeacherID_="";
9.  private String LoginName_;
10. private String JName_="";
11. private String Furigana_="";
12. private String MailAdd_="";

13. public cTeacher(){};

14. public String getTeacherID(){return this.TeacherID_;}
15. public void setTeacherID(String TeacherID){this.TeacherID_=TeacherID;}
16. public void setJName(String JName){this.JName_=JName;}
17. public String getJName(){return this.JName_;}
18. public String getLoginName() {return this.LoginName_;}
19. public void setLoginName(String name) {this.LoginName_ = name;}
20. public String getFurigana(){return this.Furigana_;}
21. public void setFurigana(String Furigana){this.Furigana_=Furigana;}
22. public String getMailAdd(){return this.MailAdd_;}
23. public void setMailAdd(String MailAdd){this.MailAdd_=MailAdd;}

24. public String CreateUpdateSQLString(){
25.    String sql="UPDATE teachers SET loginname='" + this.LoginName_ +"' jname='" +
       this.JName_ +"'furinaga='" + this.Furigana_ +"' mailadd='" + this.MailAdd_ +"'
       WHERE teacherid ='" + this.TeacherID_ + "'";
26.    return sql;
27. };

28. public String CreateDeleteSQLString(){
29.    String sql="DELETE FROM teachers WHERE ( teacherid = '" + this.TeacherID_ + "')";
30.    return sql;
31. };

32. public String CreateInsertSQLString(){
33.    ArrayList DBCol = new ArrayList() ;
34.    ArrayList Values = new ArrayList();
35.    DBCol.add("teacherid");
36.    DBCol.add("loginname");
37.    DBCol.add("jname");
38.    DBCol.add("furigana");
39.    DBCol.add("departmentdivisionid");
40.    DBCol.add("mailadd");
41.    Values.add("'"+ this.TeacherID_ + "'");
42.    Values.add("'"+ this.LoginName_ + "'");
43.    Values.add("'"+ this.JName_ + "'");
44.    Values.add("'"+ this.Furigana_ + "'");
```

```
45.   Values.add("'"+ this.divisionID_ + "'");
46.   Values.add("'"+ this.MailAdd_ + "'");
47.   String sql="INSERT INTO teachers ( teacherid, loginname, jname, furigana, mailadd )
      VALUES ( '" + this.TeacherID_ + "','" + this.LoginName + "','" + this.Jname + "','" +
      this.mailadd + "')";
48.   return sql;
49. };

50. public void getObjectFromDB(ResultSet rs){
51.   try{
52.     this.setTeacherID(rs.getString("teacherid"));
53.     this.setLoginName(rs.getString("loginname"));
54.     this.setJName(rs.getString("jname"));
55.     this.setFurigana(rs.getString("furigana"));
56.     this.setMailAdd(rs.getString("mailadd"));
57.   }
58.   catch (SQLException e) {
59.   e.printStackTrace();
60.   }
61. }

62. public void getTeacherFromID(){
63.   String sql="select * from vteacher where teacherid = '" + this.TeacherID_ + "'";
64.   this.getObject(sql);
65. }

66. public void getTeacherFromLoginName(){
67.   String sql="select * from vteacher where loginname = '" + this.LoginName_ + "'";
68.   this.getObject(sql);
69. }
70. }
```

図 15　cTeacher クラスのコード

「cTeaher クラスコードの解説」

行 1 ～行 6：　パッケージ aichi.wbces.teacher の中に、cTeacher クラスを宣言し、また、この cTeacher クラスの中に、aichi.wbces.dao パッケージと複数の java.sql、java.util と java.text パッケージの利用を宣言する。

行 7：　cDAO クラスを継承しながら、cTeacher クラスを宣言する。

行 8 ～行 12：　cTeacher クラスフィールドをカプセルの内部で宣言する。

行 13：　インスタンスの初期処理を行うためのコンストラクタ（Constructor）

行 14 ～行 23：　クラスの外部からフィールドを操作するためのメソッドを宣言する。

行 24 ～行 27：　cDAO に定義した CreateUpdateSQLString 抽象メソッドを実装する。

行 28 ～行 31：　cDAO に定義した CreateDeleteSQLString() 抽象メソッドを実装する。

行 32 ～行 49：　cDAO に定義した CreateInsertSQLString 抽象メソッドを実装する。

行 50 ～行 61：　cDAO に定義した getObjectFromDB 抽象メソッドを実装する。

行 62 ～行 65：　教員 ID から教員のデータを抽出し、教員インスタンスを作り出すメソッドを定義する。

行 66 ～行 69：　LoginName からデータを抽出し、教員インスタンスを作り出すメソッドを定義する。

抽象メソッド、CreateUpdateSQLString、CreateDeleteSQLString() と CreateInsertSQLString の実装は、カプセル化したオブジェクトの内部フィールド情報をカプセルの外部へ取り出し、テーブルのフィールドに合わせながら、それぞれ Update、Delete、Insert タイプの SQL 文を構成する。その過程はディス・カプセル（Discapsule）と呼ぶ。

一方、抽象メソッド getObjectFromDB の実装は、逆の方向でデータを流れている。教員テーブルからデータが抽出され、そのデータは ResultSet 変数 rs を通して引き渡される。次に ResultSet 変数 rs に格納している教員テーブルの各フィールドデータを、それぞれクラスフィールドに合わせ、教員オブジェクトの内部フィールドに入れる。その過程はカプセル化と呼ぶ。

カプセルとディス・カプセルの過程により、オブジェクトとデータベースの間にデータマッピングが行われ、詳細な作業は cDAO の getObject(sql) に集約されたため、getTeacherFromID() メソッドと getTeacherFromLoginName() メソッドは非常に簡潔な形で記述できた。

次に教員リストクラスへの拡張コードを解説する。

```
1.  package aichi.wbces.teacher;
2.  import java.sql.*;
3.  import java.util.*;
4.  mport aichi.wbces.dao.*;

5.  public class beanTeacherList extends cGetObjectFromDB {

6.  public beanTeacherList(){};

7.  public Object getObjectFromDB(ResultSet rs){
8.      cTeacher aTeacher = new cTeacher();
9.      aTeacher.getObjectFromDB(rs);
10.     return aTeacher;
11. }

12. public ArrayList getTeacherList(String sql){
13.      ArrayList aTeacherList = this.getObjectList(sql);
14.      return aTeacherList;
15. }
16. }
```

図 16　beanTeacherList クラスのコード

「beanTeaherList クラスコードの解説」

行 1 ～行 4：　パッケージ aichi.wbces.teacher の中に、beanTeacherList クラスを宣言し、また、このクラスの中に、aichi.wbces.dao パッケージと java.sql と java.util パッケージの利用を宣言する。

行5：　　cGetObjectFromDBクラスを継承しながら、beanTeacherListクラスを宣言する。

行6：　　インスタンスの初期処理を行うためのコンストラクタ（Constructor）である。

行7～行11：　　cGetObjectFromDBに定義したgetObjectFromDB抽象メソッドを実装する

行12～行15：　　複数の教員インスタンスを配列型で返すメソッドを定義する

抽象メソッドgetObjectFromDBの実装に、cTeacherのインスタンスが使われることで、継承しているcDAOのgetObjectFromDB(rs)がそのまま再利用できる。また、教員リストを抽出するためのメソッドgetTeacherList(sql)も、cGetObjectFromDBクラスから継承したgetObjectList(sql)メソッドを利用し、簡単に教員リストを返すことができた。

今回のオブジェクト指向の設計案が、いまの段階になって、コードの再利用性にもたらすメリットが実感できるようになった。

5. 応用事例

DAOパッケージを利用することで、サーブレットとJSPでのプログラミングは極めて簡潔になった。この節では、2つの事例を通してデータベースプログラムデザインパターンを利用したサーブレットとJSPプログラミングを紹介する。

5.1　事例1：学生の履修科目一覧を抽出し、Web上表示する

作業の流れは次のようになる。まず、学籍番号（UserID）を指定し、下図のサーブレットメソッドjobShowLectureListに送信する。サーブレットでは、データビューvtakelecture（学生履修情報・アンケート実施状態を集めるデータビュー）から、該当する履修科目リストを抽出し、その結果をJSPファイルへ送る。そしてJSPファイルは、サーブレットから送ってきたリストを受け取って、HTMLコードに埋め込む。次に作業の流れにそってコードを紹介する。

```
1. private void jobShowLectureList(HttpServletRequest request, HttpServletResponse response,
                                   String UserID) throws ServletException, IOException {

2. HttpSession session = request.getSession(true);
3. beanClassStudentList cList = new beanClassStudentList();
4. String sql="select * from vtakelecture where studentid='" + UserID + "' order by fullclassname";
5. ArrayList ClassList = cList.getClassStudentList(sql);
6. session.setAttribute("sesLectureList",ClassList);
7. response.sendRedirect("ces/students/side.jsp");
8. }
```

図17　学生履修科目リストを抽出するサーブレットのコード

コードの説明：

行1：　メソッドjobShowLectureListを定義し、学籍番号をUserIDとして受け取る。

行2：　履修科目データを次のJSPファイルへ転送するために、ウェッブアプリケーションのセッ

ションを設定する。

行3： 履修科目リストを扱うクラス beanClassStudentList のインスタンスを cList として定義する。

行4： SQL 文を記述する。

行5： SQL 文を実行し、抽出した履修科目のリストを配列 ClassList に格納する。

行6： 配列 ClassList をセッション sesLectureList に乗せて、JSP へ発信する。

行7： 次に、ブラウザに学生画面のメニューページ "ces/students/side.jsp" 表示するように指示する。

次に、JSP 側のコードを見てみよう。

```
1.  <%@ page contentType="text/html; charset=EUC-JP" session="true"
                                import="java.util.*,aichi.wbces.subject_class.*"%>
2.  <html>
3.  <head>
4.  <title> 学生 </title>
5.  </head>
6.  <% ArrayList ClassList = (ArrayList)session.getAttribute("sesLectureList");%>

7.  <table>
8.  <tr><td> 履修科目名 </td><td> 回答状況 </td></tr>
9.  <% int ListSize=ClassList.size();
10. for (int i=0; i<ListSize; i++) {
11.         cClassStudent s = (cClassStudent)ClassList.get(i); %>
12.         <tr>
13.             <td><%= s.getFullClassName() %></td><td><%= s.getAnswer() %></td>
14.         </tr>
15. <% } %>
16. </table>
17. </body>
18. </html>
```

図18　学生履修科目リストを表示する JSP のコード

コードの説明：

行1： JSP ページが呼び出されるたびに実行するディレクティブである。その中に HTML より出力、EUC_JP の文字コードの指定と java.util と aichi.wbces.subject_class パッケージの利用を宣言する。

行2～行18： HTML コード。その中、以下の JSP のプログラムは埋め込まれている。

行6： セッション sesLectureList によりサーブレットから送られっている履修科目リストデータを受け取り、配列 ClassList に格納する。

行9： 配列 ClassList のサイズで、履修科目の数を取得する。

行10～行14： 一科目一行で、HP 上に表を書き出す。そのうち、

行11： 配列から各々の履修科目を cClassStudent クラスのインスタンス s に格納する

行13： スクリプト式 <% = %> で、履修科目名 s.getFullClassName() とアンケート実

施状態 s.getAnswer() をテーブルのセルに埋め込む

データベースの外部で、各々のクラス内部で記述した SQL 文を、DAO パッケージを通して、データベースを操作する仕組みをオブジェクト指向のデザインパターンで実現した。

ところが、より複雑なデータベースの操作、或いは大量なデータベース操作をまとめて行う場合、データベース内部の Store Procedure を利用する必要がある。Store Procedure とは、データベース操作ためのプログラムであり、データベース内部に置くことで、効率的に実行することができる。

DAO パッケージを Store Procedure の使用まで拡張することは、WBCES システムにとって非常に重要である。次には、管理員のアンケート初期設定作業を例として紹介し、DAO の Store Procedure への使用を説明する。

5.2 事例 2：後期アンケートの初期設定

作業には以下 3 つの目的がある。

(1) 春・秋学期ごとに、後期アンケートの実施日を決める

(2) 全ての開講科目を対象に、1 回だけの後期アンケートを設定する

(3) 以上のデータを一括的に inquiry テーブルに書き込み

一回でやく 2000 個以上のデータを一気に書き込まれるので、そのために以下の Store Procedure プログラム insert_final_inquiry が使われている。

```
1. declare
2. term_no alias for $1;
3. start_date alias for $2;
4. total integer;
5. begin
6. execute 'insert into inquiry(classid, termno, inquiryno, startdate, question1,
   question2, comment1, comment2, resultpath, publish)
   select classid, term_no as termno, 12 as inquiryno, start_date as inquirydate,
   '' as question1, ''as question2, ''as comment1, ''as comment2,
   '' as resultpath, -1 as publish
   from classes where termno= term_no and classid not in
   (select classid from inquiry where termno= term_no)';[1]
7. get diagnostics total = row_count;
8. return total;
9. end;
```

図 19　アンケート初期設定ための Store Procedure コード：insert_final_inquiry

コードの説明：

行 1～行 4：　引数 term_no（学期番号）、start_date（開始日）を宣言する。
また、データベースの処理回数を結果として返すための変数 total を宣言する。

行 6：　inquiry テーブルへデータを追加するための SQL 実行文。

[1] この SQL 文を読みやすくするために、いくつかの文字処理ための記号を省略している。

行7：　　データベースの処理回数を取得する。

行8：　　データベースの処理回数を返す。

図19に示されたStore Procedureコードを実行するためには、cDAOクラスの中に下図に示すexecuteSQLstatementメソッドを用意した。

```
1. public void executeSQLstatement(String sql) {
2.    Connection cn=this.createConnection();

3.    try {
4.         Statement st = cn.createStatement();
5.         st.execute(sql);
6.    }
7.    catch (Exception e) {
8.         e.printStackTrace();
9.    } finally {
10.         this.closeConnection(cn);
11.         }
12.   }
13. }
```

図20　cDAOクラスのexecuteSQLstatementメソッド

そこで第5行目のst.execute(sql)は、JDBCパッケージのStatementオブジェクトのメソッドである。実行するStore Procedureの指定は、sql文に記述し、st.execute(sql)の形で実行させる。

この作業は管理員の権限であり、クラスcManagerがcDAOを継承することで、管理員はこのStore Procedureを駆動するメソッドが利用できる。

以下に、サーブレットに後期アンケートの初期設定ためのjobManagerSetFinalInquiryメソッドを通して、executeSQLstatementの使い方を説明する。

```
1.  private void jobManagerSetFinalInquiry(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

2.  cManager manager = new cManager();

3.  int TermNo = Integer.parseInt(request.getParameter("term_no"));
4.  int year1 = Integer.parseInt(request.getParameter("year"));
5.  int month1=Integer.parseInt(request.getParameter("month"));
6.  int day1 = Integer.parseInt(request.getParameter("day"));

7.  String strDate = year1 + "-" + month1 + "-" + day1;
8.  String sql="select insert_final_inquiry_date(" + ts.getTermNo() + ",'" + strDate + "')";

9.  manager.executeSQLstatement(sql);
10. getServletConfig().getServletContext().getRequestDispatcher("/ces/managers/project.jsp").
    forward(request,response);
11. }
```

図21　cDAOクラスのexecuteSQLstatementメソッド

コードの説明：

行1：　　メソッドjobManagerSetFinalInquiryを定義する。

行2：　　管理員クラスのインスタンス manage を定義する

行3～行6：　　外部 HP から学期番号、アンケート開始日の年、月と日を取得する

行7：　　アンケート開始日の文字列を作成。

行8：　　Store Procedure 関数 insert_final_inquiry を指定するための SQL 文を記述する。

行9：　　executeSQLstatement(sql) で Store Procedure を駆動する。

行10：　　ブラウザが管理員画面に戻る。

まとめ

大学の授業評価システム WBCES の開発にあたって、オブジェクト指向における再利用のためのデータベースプログラミングデザインパターンを提案した。このデザインパターンは、Java データベースプログラムで利用される JDBC パッケージに関連するコードと、各々のオブジェクトの特有な SQL クエリを分離させることで、高いコードの再利用性が得られた。

WBCES の関連事例を紹介することにより、提案したデータベースプログラミングのデザインパターンは、高度な汎用性を持って、広範囲の Java ウェブアプリケーションに応用できることが判明した。

参考文献

[1] アイティーブースト著、はじめての JSP &サーブレットプログラミング、秀和システム、2004.

[2] 今野睦　等著、サーブレット /JSP、ソフトバンク社、2003.

[3] 今野睦　等著、データベース設計のための UML、翔泳社、2003.

[4] 結城浩著、オブジェクト指向における再利用のためのデザインパターン、ソフトバンク社、2002.

[5] Richard Stones、Neil Matthew 著、エキスパートから学ぶ PostgreSQL 活用テクニック、インプレス社、2002.