上級DAOプログラミング

より良いDAO構築の技法を学ぶ

Sean Sullivan (dao-article@seansullivan.com)
Software Engineer

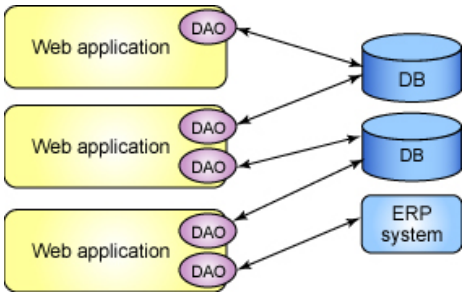
2003年 10月 07日

J2EE開発者はデータ・アクセス・オブジェクト (DAO)デザインパターンを使って、低レベルのデータ・アクセスを高レベルのビジネス・ロジックから分離します。DAOパターンの実装はデータ・アクセス・コードを書く以上のものが関係してきます。この記事ではJava開発者である、Sean C. SullivanがDAOプログラミングで見過ごされがちな3つの面、トランザクション境界設定、例外処理、ロギングについて説明します。

ここ18ヶ月の間、私は技術者チームの一員としてカスタム仕様のWebベースでのサプライチェーン・マネージメントの構築に携わってきました。このアプリケーションは非常に広範囲で永続性のあるデータを扱うもので、発送状況、サプライチェーンの性能評価指数 (supply chain metrics)、在庫、輸送業者の請求書、プロジェクト・マネージメントのデータ、それにユーザーのプロファイルなどを対象にしています。我々の会社の各種データベース・プラットフォームへの接続にはJDBC API を使いましたが、このアプリケーションには一貫してDAOデザインパターンを採用しました。

図1はアプリケーションとデータソースの関係を示します。

図1. アプリケーションとデータソース



アプリケーション全体にデータ・アクセス・オブジェクト (DAO) パターンを採用したおかげで、低レベルデータ・アクセス・ロジックをビジネス・ロジックから分離することができました。また我々は、各データソースに対してCRUD (create, read, update, delete)操作を提供するDAOクラスを構築しました。

この記事ではDAO実装への取り組み方、より良いDAOクラスを構築するテクニックについて説明します。具体的にはロギング、例外処理、トランザクション境界設定を中心に説明しますので、この3つすべてをどうやってDAOクラスに取り入れるかが分かるでしょう。この記事では読者がJDBC API、SQL、それにリレーショナル・データベースは良く分かっていると言う前提でお話します。

最初にDAOデザインパターンとデータ・アクセス・オブジェクトの復習から始めましょう。

DAOの基礎

DAOデザインパターンは標準のJ2EEデザインパターンの一つです。開発者は低レベルのデータ・アクセス操作を高レベルのビジネス・ロジックから分離するのにこのパターンを使用します。典型的なDAO実装には次のコンポーネントがあります。

- DAOファクトリー・クラス
- DAOインターフェース
- DAOインターフェースを実装するコンクリートクラス
- データ転送オブジェクト（バリュー・オブジェクトと言われることもあります）

コンクリートDAOクラスは、ある特定のデータソースからデータをアクセスするロジックを含んでいます。後ほど、データ・アクセス・オブジェクトの設計と実装のテクニックを説明します。DAOデザインパターンについての詳細は参考文献を見てください。

トランザクション境界設定

DAOで頭に置いておくべきなのは、DAOがトランザクショナルなオブジェクトであるということです。DAOの各操作、例えばデータの生成、更新、削除などの各操作は、ある一つのトランザクションと関連付けられています。ですからトランザクション境界設定 の概念は非常に重要なのです。

トランザクション境界というのはトランザクションの境界定義の仕方を言います。J2EE仕様ではトランザクション境界設定として2つのモデル、プログラマチック・トランザクション境界設定と宣言トランザクション境界設定を記述しています。表1はこの2つの違いです。

表1. トランザクション境界設定の2つのモデル

宣言トランザクション境界設定	プログラマチック・トランザクション境界設定
プログラマーがEJBデプロイメント・デスクリプタを使ってトランザクション属性を宣言する	プログラマーがトランザクション・ロジックのコーディングに責任を持つ
ランタイム環境（EJBコンテナ）が属性を使ってトランザクションを自動管理する	アプリケーションがAPIでトランザクションを制御する

ここではプログラマチック・トランザクション境界設定に焦点を絞ります。

設計にあたって考慮すべきこと

前に述べた通り、DAOはトランザクショナルなオブジェクトです。典型的なDAOはトランザクショナルな操作、つまり生成、更新、削除などを行います。DAOを設計するに当たってはまず、次のような点について自問してみてください。

- トランザクションはどのように開始されるか？
- トランザクションはどのように終了するか？
- どのオブジェクトがトランザクションを開始することになっているか？
- どのオブジェクトがトランザクションを終了することになっているか？
- DAOがトランザクションを開始・終了することになっているか？
- アプリケーションは複数のDAOにまたがってデータをアクセスする必要があるか？
- 一つのトランザクションは一つのDAOのみを含むのか複数のDAOか？
- DAOは他のDAOのメソッドを起動するか？

こうした質問に対する答えが分かっていれば、自分のDAOに一番合った形での、トランザクション境界設定への取り組み方を選びやすくなります。DAOにおいてトランザクション境界設定に取り組むには2つの方法があります。一つはDAOにトランザクションの境界設定を任せる方法、もう一つはトランザクションの境界設定をDAOのメソッドをコールしているオブジェクトに任せる方法です。前者ではトランザクション・コードをDAOクラスに埋め込みます。後者ではトランザクション境界設定コードはDAOクラスの外になります。簡単なコード例を使って、この２つの方法がどのように動作するのかを説明します。

リスト1は2つのデータ操作、生成と更新のあるDAOを示します。

リスト1. DAOメソッド

```
public void createWarehouseProfile(WHProfile profile);
public void updateWarehouseStatus(WHIdentifier id, StatusInfo status);
```

リスト2は簡単なトランザクションを示します。トランザクション境界設定コードはDAOクラスの外です。この例では呼び出し元がトランザクション内で、どのように複数のDAO操作を組み合わせているかに注意してください。

リスト2. 呼び出し元管理のトランザクション

```
tx.begin();    // start the transaction
dao.createWarehouseProfile(profile);
dao.updateWarehouseStatus(id1, status1);
dao.updateWarehouseStatus(id2, status2);
tx.commit();   // end the transaction
```

このトランザクション境界設定の方法は、一つのトランザクションで複数のDAOにアクセスする必要のあるアプリケーションには特に重要と言えます。

トランザクション境界設定はJDBC APIまたはJava トランザクション API (JTA)いずれかを使って実装できます。JDBCトランザクション境界設定はJTAトランザクション境界設定よりも簡単ですが、JTAの方が柔軟性に優れています。後の章ではトランザクション境界設定の機構をもう少し詳しく見ていくことにします。

JDBCによるトランザクション境界設定

JDBCトランザクションは`Connection`オブジェクトを使って制御されます。JDBCコネクション・インターフェース（`java.sql.Connection`）には2つのトランザクション・モジュール（自動コミットと手動コミット）が用意されています。`java.sql.Connection`にはトランザクション制御に関して次のようなメソッドがあります。

- `public void setAutoCommit(boolean)`
- `public boolean getAutoCommit()`
- `public void commit()`
- `public void rollback()`

リスト3はJDBC APIを使用して、トランザクションをどのように境界設定するかを示します。

リスト3. JDBC APIによるトランザクション境界設定

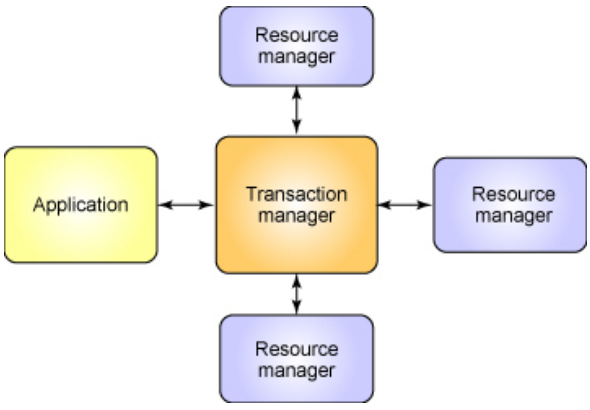
```
import java.sql.*;
import javax.sql.*;
// ...
DataSource ds = obtainDataSource();
Connection conn = ds.getConnection();
conn.setAutoCommit(false);
// ...
pstmt = conn.prepareStatement("UPDATE MOVIES ...");
pstmt.setString(1, "The Great Escape");
pstmt.executeUpdate();
// ...
conn.commit();
// ...
```

JDBCトランザクション境界設定により、複数のSQLステートメントを組み合わせて一つのトランザクションにすることができます。JDBCトランザクションの欠点の一つは、トランザクションのスコープが一つのデータベース・コネクションに限られるという点です。JDBCトランザクションは複数のデータベースにまたがることができないのです。次に、JTAを使ったトランザクション境界設定がどのように行われるかを見てみます。JTAはJDBCほど知られていないので、JTAの概略から始めることにします。

JTAの概略

JavaトランザクションAPI（JTA）とその兄弟分、Javaトランザクション・サービス（JTS）はJ2EEプラットフォーム用の分散トランザクション・サービスを提供します。分散トランザクション にはトランザクション・マネージャーと、一つまたは複数のリソース・マネージャーが関係します。リソース・マネージャー は任意の種類の永続的なデータソースです。トランザクション・マネージャーはトランザクションに関係するもの同士のコミュニケーションを調整します。トランザクション・マネージャーとリソース・マネージャーの関係を図2に示します。

図2. トランザクション・マネージャーとリソース・マネージャー



JTAトランザクションはJDBCトランザクションよりもずっと強力です。JDBCトランザクションでは一つのデータベース・コネクションに制限されていますが、JTAトランザクションは複数の参加者を持つことができます。次に示すJavaプラットフォームのコンポーネントはどれでも、このうちの一つがJTAトランザクションに参加することができます。

- JDBCコネクション
- JDBC`PersistenceManager`オブジェクト
- JMSキュー
- JMSトピック
- エンタープライズJavaビーンズ
- J2EEコネクター・アーキテクチャー仕様に合致したリソース・アダプター

JTAによるトランザクション境界設定

JTAでトランザクション境界設定をするには、アプリケーションは`javax.transaction.UserTransaction`インターフェースのメソッドを起動します。リスト4は、`UserTransaction`オブジェクトへの典型的なJNDIルックアップを示します。

リスト4. UserTransactionオブジェクトへのJNDIルックアップ

```
import javax.transaction.*;
import javax.naming.*;
// ...
InitialContext ctx = new InitialContext();
Object txObj = ctx.lookup("java:comp/UserTransaction");
UserTransaction utx = (UserTransaction) txObj;
```

`UserTransaction`オブジェクトへの参照ができると、アプリケーションはトランザクションを開始します。これをリスト5に示します。

リスト5. JTAでトランザクションを開始する

```
utx.begin();
// ...
DataSource ds = obtainXADataSource();
Connection conn = ds.getConnection();
pstmt = conn.prepareStatement("UPDATE MOVIES ...");
pstmt.setString(1, "Spinal Tap");
pstmt.executeUpdate();
// ...
utx.commit();
// ...
```

アプリケーションが`commit()`を起動すると、トランザクション・マネージャーは2フェーズ・コミット・プロトコルを使ってトランザクションを終了します。

トランザクション制御用のJTAメソッド

`javax.transaction.UserTransaction`インターフェースには次のようなトランザクション制御メソッドがあります。

- `public void begin()`

- `public void commit()`
- `public void rollback()`
- `public int getStatus()`
- `public void setRollbackOnly()`
- `public void setTransactionTimeout(int)`

アプリケーションはトランザクションを開始するのに`begin()`をコールします。トランザクションを終了するには`commit()`または`rollback()`をコールします。JTAでのトランザクション管理についての詳細は参考文献を見てください。

JTAとJDBCを使う

開発者はDAOクラスでの低レベルのデータ操作に、よくJDBCを使います。JTAでトランザクション境界設定をするつもりならば、`javax.sql.XADataSource`と`javax.sql.XAConnection`、それに`javax.sql.XAResource`インターフェースを実装した JDBCドライバーが必要です。こうしたインターフェースを実装したドライバーであればJTAトランザクションに参加できます。`XADataSource`オブジェクトは`XAConnection`オブジェクトのファクトリーです。`XAConnections`はJTAトランザクションに参加するJDBCコネクションです。

ここではアプリケーション・サーバーの管理ツールを使って`XADataSource`を設定する必要があります。その具体的な方法についてはアプリケーション・サーバーやJDBCドライバーの資料を参照してください。

J2EEアプリケーションはJNDIを使ってデータソースをルックアップします。一旦データソース・オブジェクトへの参照ができれば、アプリケーションはデータベースに接続できるように、`javax.sql.DataSource.getConnection()`をコールします。

XAコネクションは非XAコネクションと異なります。XAコネクションはJTAトランザクションに参加していることは常に頭に置いておいてください。つまり、XAコネクションはJDBCの自動コミット・フィーチャーはサポートしていないということです。また、アプリケーションはXAコネクション上の`java.sql.Connection.commit()`や`java.sql.Connection.rollback()`を起動してはならず、代わりに`UserTransaction.begin()`と`UserTransaction.commit()`、それに`UserTransaction.rollback()`を使うことになっています。

最良の方法を選択する

トランザクション境界設定をJDBCで行う方法とJTAで行う方法の両方を説明してきました。どちらにも利点があり、どちらが自分のアプリケーションに適切かを考えて選択する必要があります。

最近携わった多くのプロジェクトで、我々のチームはトランザクション境界設定にJDBC APIを使ってDAOクラスを構築しました。このDAOクラスは次のように要約できます。

- トランザクション境界設定コードはDAOクラス内に埋め込まれている
- DAOクラスはトランザクション境界設定にJDBC APIを使っている
- 呼び出し元ではトランザクション境界設定できない
- トランザクション・スコープは単一のJDBCコネクションに限られる

JDBCトランザクションは、企業レベルの複雑なアプリケーションには必ずしも向いていません。トランザクションが複数のDAOやデータベースにまたがる場合は、次のような実装方針がより適切と言えます。

- トランザクションはJTAで境界設定する
- トランザクション境界設定はDAOと分離する
- 呼び出し元がトランザクション境界設定を受け持つ
- DAOはグローバル・トランザクションに参加する

JDBCを使った方法は簡単ですので魅力があります。またJTAでの方法は柔軟性に優れています。どちらを選ぶかはアプリケーション固有の要件次第と言えます。

ロギングとDAO

うまく実装できたDAOクラスでは、ランタイムの振る舞いの詳細をキャプチャーするのにロギングを使います。ログする対象は選択でき、例外、コンフィグレーション情報、コネクション状態、JDBCドライバーのメタデータ、クエリー・パラメータなどが対象になります。開発のどの段階でもログは利用価値のあるものです。私は開発途中やテスト中、それに製品化段階でも頻繁にアプリケーションのログを調べます。

この章ではJakarta Commons Logging をどうやってDAOに適用するか、コード例を示しながら説明します。その前に、ちょっと基本を復習しましょう。

ロギング・ライブラリを選ぶ

開発者はよく基本形式のロギング、`System.out.println`と`System.err.println`を使います。`Println`ステートメントは手軽ですが、本格的なロギング・システムのように強力ではありません。表2にJavaプラットフォームでのロギング・ライブラリを示します。

表2. Javaプラットフォームでのロギング・ライブラリ

ロギング・ライブラリ	オープンソースか？	URL
java.util.logging	No	http://java.sun.com/j2se/
Jakarta Log4j	Yes	http://jakarta.apache.org/log4j/
Jakarta Commons Logging	Yes	http://jakarta.apache.org/commons/logging.html

`java.util.logging`はJ2SE 1.4プラットフォームでの標準APIです。ただ、大部分の開発者は私と同意見だと思いますが、Jakarta Log4jの方がずっと機能豊富でかつ柔軟性に富んでいます。J2SE 1.3とJ2SE 1.4両方のプラットフォームをサポートしていることはJakarta Log4jがjava.util.loggingよりも優れている点の一つと言えます。

Jakarta Commons Logging は`java.util.logging`やJakarta Log4jと一緒に使うことができます。Commons Logging はアプリケーションを、その下にあるロギング実装から分離するロギング抽象化レイヤーです。Commons Logging ではコンフィギュレーション・ファイルを変更することで、下にあるロギング実装を入れ替えることができます。Commons Logging はJakarta Struts 1.1とJakarta HttpClient 2.0で使われています。

ロギング例

リスト7はJakarta Commons Logging をDAOクラスでどう使うかを示しています。

リスト7. DAOクラスでのJakarta Commons Logging

```
import org.apache.commons.logging.*;
class DocumentDAOImpl implements DocumentDAO
{
      static private final Log log = LogFactory.getLog(DocumentDAOImpl.class);
      public void deleteDocument(String id)
      {
          // ...
          log.debug("deleting document: " + id);
          // ...
          try
          {

              // ... data operations ...
          }
          catch (SomeException ex)
          {
              log.error("Unable to delete document", ex);
              // ... handle the exception ...
  }
      }
}
```

失敗が許されないようなアプリケーションでは、ロギングは最も重要な部分と言うことができます。DAOに障害があった場合、何がおかしくなったかを知るのにログの情報が一番確実です。DAOにロギングを組み込んでおくことで、デバッグやトラブルシューティングにも対応できます。

DAOでの例外処理

ここまでトランザクション境界設定とロギングを見てきて、それらがデータ・アクセス・オブジェクトにどう適用できるか理解できたと思います。3番目で最後になりますが、例外処理を取り上げます。例外処理に関して、ちょっとした指針を守ることでDAOは使いやすく堅牢に、かつ保守管理しやすくなります。

DAOパターンを実装するに当たっては次のような質問を考慮してください。

- DAOのパブリック・インターフェースはチェックされた例外をスローするか？
- もしイエスであれば、どんなものがチェックされた例外としてスローされるか？
- DAO実装クラス内で、例外はどのように処理されるか？

我々のチームでは、DAOパターンにとりかかっている間、例外処理に関して指針を作りました。この指針を守ることでDAOを大いに改善することができます。

- DAOメソッドは意味のある例外をスローすべきである。
- DAOメソッドは`java.lang.Exception`をスローしてはならない。`java.lang.Exception`は汎用すぎ、下にある問題について何の情報ももたらさない。
- DAOメソッドは`java.sql.SQLException`をスローしてはならない。SQLExceptionは低レベルのJDBC例外です。DAOはアプリケーションの他の部分に対してJDBCを見えるようにせず、むしろ積極的にカプセル化すべきである。

- DAOのメソッドは呼び出し元が例外処理することが十分想定できる場合にのみ、チェックされた例外をスローすべきである。呼び出し元が妥当な手段で例外を処理できそうもないときには、チェックされていない（ランタイムの）例外をスローすることを考慮する。
- データアクセスコードが例外を捕らえたら、それを無視しないこと。例外を捕らえておいて処理しないDAOをトラブルシューティングするのは困難です。
- 低レベルの例外を高レベルの例外に変換するにはchained exceptionsを使う。
- 標準DAO例外クラスを定義することを考える。Spring Framework（参考文献）にはあらかじめ定義された、一式の素晴らしいDAO例外クラスがあります。

例外と例外処理のテクニックの詳細については参考文献を参照してください。

実装例: MovieDAO

`MovieDAO`はこの記事で説明したテクニックのすべて、つまりトランザクション境界設定、ロギング、例外処理を説明するDAOです。`MovieDAO`のソースは参考文献にありますが、コードは3つのパッケージに分割されています。

- `daoexamples.exception`
- `daoexamples.movie`
- `daoexamples.moviedemo`

このDAOパターン実装は下記のクラスやインターフェースから成り立っています。

- `daoexamples.movie.MovieDAOFactory`
- `daoexamples.movie.MovieDAO`
- `daoexamples.movie.MovieDAOImpl`
- `daoexamples.movie.MovieDAOImplJTA`
- `daoexamples.movie.Movie`
- `daoexamples.movie.MovieImpl`
- `daoexamples.movie.MovieNotFoundException`
- `daoexamples.movie.MovieUtil`

`MovieDAO`インターフェースはDAOのデータ操作を定義しており、下記5つのメソッドを持っています。

- `public Movie findMovieById(String id)`
- `public java.util.Collection findMoviesByYear(String year)`
- `public void deleteMovie(String id)`
- `public Movie createMovie(String rating, String year, String, title)`
- `public void updateMovie(String id, String rating, String year, String title)`

`daoexamples.movie`パッケージには`MovieDAO`インターフェースを2種類実装していますが、トランザクション境界設定には異なった手法をとっています。表3にこれを示します。

表3. MovieDAO実装

	MovieDAOImpl	MovieDAOImplJTA

MovieDAOを実装しているか？	Yes	Yes
DataSourceをJNDIからとるか	Yes	Yes
java.sql.ConnectionオブジェクトをDataSourceからとるか？	Yes	Yes
DAOが内部的にトランザクション境界設定をするか？	Yes	No
JDBCトランザクションを使うか？	Yes	No
XA DataSourceを使うか？	No	Yes
JTAトランザクションに参加するか？	No	Yes

MovieDAOデモ・アプリケーション

デモのアプリケーションは`daoexamples.moviedemo.DemoServlet`と呼ばれるサーブレット・クラスです。`DemoServlet`は両方のMovie DAOを使って、表にあるムービー・データのクエリーと更新します。

このサーブレットはJTA対応の`MovieDAO`とJavaメッセージサービスを、どうやって一つのトランザクションで結合するかを説明しています。これをリスト8に示します。

リスト8. MovieDAOとJMSコードを一つのトランザクションで結合する

```
UserTransaction utx = MovieUtil.getUserTransaction();
utx.begin();
batman = dao.createMovie("R",
    "2008",
    "Batman Reloaded");
publisher = new MessagePublisher();
publisher.publishTextMessage("I'll be back");
dao.updateMovie(topgun.getId(),
    "PG-13",
    topgun.getReleaseYear(),
    topgun.getTitle());
dao.deleteMovie(legallyblonde.getId());
utx.commit();
```

このデモ・アプリケーションを走らせるには、アプリケーション・サーバーにあるXAデータソースと非XAデータソースを環境設定します。次にdaoexamples.earファイルを展開します。このアプリケーションは、J2EE 1.3準拠のアプリケーション・サーバーであれば、どれでも走らせることができます。EARファイルやソースコードの入手については参考文献を見てください。

まとめ

この記事で説明してきたように、DAOパターンの実装には単に低レベルのデータ・アクセス・コードを書く以上のことが要求されます。アプリケーションに合ったトランザクション境界設定を選び、DAOクラスにロギングを組み込み、また例外処理に関して、簡単なものですがいくつかの指針に従うことで、今日からでもより良いDAOを構築することができるでしょう。

著者について

Sean Sullivan

Sean C. Sullivanはオレゴン州ポートランドに勤務するソフトウェア技術者。一番最近ではサプライチェーン・マネージメント・アプリケーションやインターネットでのeコマース支払いシステムの構築プロジェクトに携わりました。また、IBMとImage Systems Technologyで、オペレーティング・システムやCADソフトウェアのプロジェクトにも従事したことがあります。Apache Jakarta開発者でもあり、Jakarta HttpClientプロジェクトにコードを提供しました。1996年からJavaを使ったアプリケーションを開発しており、John Wiley & Sons刊行のProgramming with the Java Media Frameworkの著者です。Rensselaer大学にてコンピューター・サイエンスで学位を取得。連絡先はdao-article@seansullivan.com。