JSPおよびJDBCテクノロジーによる動的Webベース・データ・アクセス

Noel Bergman

2001年 9月 01日

この記事では、JSPおよびJDBCテクノロジーを用いて、Webサイトの静的、動的、さらにデータベースのコンテンツを統合する方法について説明します。簡潔に実例で説明するため、この記事のJSPページでは、短いスクリプトレットを使います。これは基礎となるJDBC概念をカスタム・タグの中に隠すのではなく、JSP開発者の目にふれるようにするためです。著者は、JavaBeansコンポーネントをJDBCに統合する主要設計アプローチを紹介しますが、これはJavaServer Pagesテクノロジーで既にHTTPにBeansを使用している方法とよく似たものです。さらに、この統合を実装するためのコードも掲載しています。

Javaサーブレット・テクノロジーを基盤とするJavaServer Pages (JSP) テクノロジーは、動的コンテンツを生成するためのコア・サーバーサイドJavaアーキテクチャーです。動的コンテンツの1つのソースは、リレーショナル・データベースです。オンライン・コミュニティーからe-コマース・トランザクションに至るあらゆるものを管理するために、Webサイトではリレーショナル・データベースを使って、カタログ項目、イメージ、テキスト、登録メンバーに関するデータなど、あらゆる種類の情報を保管します。この記事では、Java Database Connectivity (JDBC) 経由のリレーショナル・データベースに対するJSPテクノロジーのアプリケーションを扱います。JDBCは、Javaプログラムがリレーショナル・データベースと連係する手段です。

この記事を最大限に活用するには、JDBCおよびSQLに通じている必要があります。

JDBCの基礎

JDBCは、JavaコードとSQLデータベース間の橋渡しになります。プライマリーJDBCオブジェクトは、データベースへのコネクションと、そのコネクションを使って実行されるステートメントを表します。リレーショナル・データベースと共に使用される2種類の基本ステートメントは、query とupdate です。それぞれの前提条件として、まずデータベースへのコネクションを確立する必要がありますが、これは `java.sql.DriverManager` クラスで行われます。コネクションは、確立するのに長い時間 (コンピューター時間で) を要しますので、Webサーバーのようなトランザクション集約型の環境においては、可能な限りコネクションを再利用したいと考えるものです。このような再利用はコネクション・プーリングと呼ばれます。

JDBCスキルが少しばかり鈍っているというのであれば、リスト1を参照ください。このコードの断片では、テスト・データベースとのコネクションを確立し、そのコネクションで使用するステー

トメント・オブジェクトを作成し、SQL照会を発行し、結果を処理し、JDBCリソースを解放する方法を説明しています。

リスト1. 簡単なJDBCコード

```
Connection connection = DriverManager.getConnection(URL, user, password);
Statement statement = connection.createStatement();
ResultSet results = statement.executeQuery(sqlQuery);

while (results.next())
{
    ... process query results ...
    logSQLWarnings(results.getWarnings());
}

results.close();
statement.close();
connection.close();
```

実際には、JDBCコードはこのように簡単ではありません。例外と警告の条件を処理する必要があります。 リスト2 は、同じJDBCの例を表していますが、JDBC例外と警告の処理を加えてあります。この例では、例外と警告は単にログに記録され、例外が発生した場合は、オペレーションを異常終了させます。ただし、 `finally{}` 文節で、必ずリソースのクリーンアップが進められるようにします。

結果の実際の処理は、ここでは暗に示されているだけです。このことについては、この記事の後半でさらに詳しく検討することにします。照会ではなくデータベースの更新を行っていたとしたら、 `while` ループを次のように置き換えることになったでしょう。

```
int count = statement.executeUpdate(sqlUpdate);
```

`executeQuery()` と `executeUpdate()` に加えて、 `Statement` クラスは一般的な `execute()` メソッドをサポートします。このため一般的なSQLコードのオーサリングが可能になりますが、結果を処理することは一層複雑な作業になります。

`executeUpdate()` メソッドは、 `update` ステートメントで更新された行数を返します。

もしもこれらのコード・リストで取り扱っている内容に不案内であれば、 参考文献 に紹介されているJDBCチュートリアル情報のいくつかをご覧になることをお薦めします。

JSPページでJDBCを使う

では、どのようにしてJDBCおよびJSPテクノロジーを組み合わせて、動的コンテンツがデータベースから送られるようにするのでしょうか?

一般に、JSPにおける実践法では、プレゼンテーションをモデルの振る舞いから分離するよう勧めています。これは、オブジェクト指向プログラミングにおけるModel-View-Controller (MVC) パラダイム と類似しています。分離する理由の1つとして、JSPテクノロジーをベースとするアプリケーションでは、ModelとControllerのコンポーネントはプログラマーによって作成され、Viewコンポーネントはページ設計者によって作成される可能性が高いということがあげられます。JSPアプリケーション・アーキテクチャーの場合、プレゼンテーションを担当しているViewの役割はJSPページによって処理されます。要求への応答を担当しているControllerの役割は、サーブレットに

よって果たされる場合がほとんどですが、多くのJSPの専門家はControllerの役割にJSPページを利用することのメリットを認識するようになっています。アプリケーション・エンティティーの振る舞いのモデル化を担当しているModelの役割は通常、JavaBeansコンポーネントによって果たされます。

MVCパラダイムのどこでデータベースと対話を行うか判断することに加え、JDBCテクノロジーをJSPページに統合するための選択肢がいくつかあります。たとえば、スクリプトレットを使ってJDBCを挿入することも、タグ・ライブラリーを使ってこれを挿入することも、あるいはこれをカスタム・タグまたは他のクラス内に隠すこともできます。次に、アプローチの例をいくつか検討し、その使用について説明します。

JSPスクリプトレットの例

新参のJSPプログラマーが手始めによく手がけることは、JDBCにアクセスするスクリプトレットを作成することです。おそらくそれは、JDBCを使ってページの「ヒット・カウンター」を実装する以下のリスト3の例のようなものになるでしょう。

リスト3. スクリプトレットにJDBCを利用するJSPページ

```
<jsp:directive.page import="java.sql.*" />
<jsp:scriptlet>
Class.forName("org.gjt.mm.mysql.Driver");
Connection connection =
    DriverManager.getConnection("jdbc:mysql://localhost:3306/test", ", ");
Statement statement = connection.createStatement();
int changed =
    statement.executeUpdate("update counters set hitCount = hitCount + 1 " +
                "where page like '" + request.getRequestURI() + "'");
if (changed == 0) statement.executeUpdate("insert counters(page) values('" +
                        request.getRequestURI() + "')");
ResultSet rs =
    statement.executeQuery("select hitCount from counters where page like '" +
                request.getRequestURI() + "'");
rs.next();
int hitCount = rs.getInt(1);
statement.close();
connection.close();
</jsp:scriptlet>

<HTML>
<HEAD>
<TITLE>JDBC scriptlet example</TITLE>
</HEAD>
<BODY>

<P>This page has been hit
<jsp:expression>hitCount</jsp:expression>
times. The page counter is implemented by a JSP scriptlet calling
the JDBC package directly.</P>

</BODY>
</HTML>
```

このページには、内部にスクリプトレットがありますが (最初のハイライト表示のセクション)、それがデータベースに接続し、ステートメントを作成し、ページのURIをキーとするカウンター・レコードを更新します。更新によってどの行も変更されない場合には、この例はそのようなレコードがないものとみなし、レコードを１行を加えます。最後に、スクリプトレットは現在の

ヒット・カウントをデータベースに照会し、その結果をローカル変数に割り当てます。さらに下って、このJSPページの「presentation」部分では、ハイライト表示されているJSP式が使用され、ヒット・カウンターの値を提示します。

> **注 :**
> 実際には、ヒット・カウンターをこのように実装しようとはしないはずです。要求ごとにデータベースを更新するコストは、不要な支出です。ただし、このヒット・カウンターはデータベースの更新と照会を行う簡単な例を示してくれるので、JSPページをJDBCで統合するさまざまな方法を説明する際に利用できます。

確かに十分機能するのですが、このJSPにはいくつか問題があります。まず、スクリプトレットは、プログラミングを行わないページ設計者が、ページに組み込もうとするものではないという点です。率直に言うと、これは実際のページのコンテンツを取り散らかしているので、プログラマーとしてもページに組み込もうと思う代物でないのです。第2は、例を簡単なものにしておくため、このページには実際のJDBCコードが必ず備えているはずの例外処理が欠けているという点です。第3は、ヒット・カウンターの実装が文字列としてJSPページに組み込まれているので、ヒット・カウンターに変更があれば、ヒット・カウンターを維持するすべてのJSPページに伝える必要が生じるという点です。

では、どのようにすれば、このJSPページ を修正できるでしょうか? 大いにもてはやされたソリューションの1つは、タグ・ライブラリーを用いてスクリプトレットを除去することです。次の例では、この代替ソリューションを検討してみます。

DBTagを利用するタグ・ライブラリーの例

新参のJSPプログラマーが、通常好意的な友人や専門家から聞かされる最初のアドバイスの1つは、スクリプトレットを使わないようにする、ということです。その代わりに、新参のJSPプログラマーはカスタム・タグを使うよう教えられます。カスタム・タグは、JSPプラットフォームの能力を拡張する手段です。コード・ライブラリーに関連づけられているカスタムのXMLスタイル・タグは、望みどおりの機能性を実装します。それがいかにうまく機能するのか、次の例で確かめてみましょう。

Jakarta TagLibs ProjectはJakarta Project (参考文献 を参照) のサブプロジェクトで、Jakartaプロジェクトとは、Java ServletおよびJavaServer Pagesテクノロジーの公式な参照インプリメンテーションのことです。

Jakarta TagLibs Projectの後援を受けて開発されたパッケージの1つには、DBTagsカスタム・タグ・ライブラリー (以前はJDBCタグ・ライブラリーとして知られていたもの) があります。リスト4のJSPページは、リスト3にあるものと同様のヒット・カウンターを実装し、スクリプトレットをカスタム・タグと置き換えます。

リスト4. DBTagを利用するJSPページ

```
<HTML>
<HEAD>
<TITLE>Jakarta DBTags example</TITLE>
</HEAD>
<BODY>

<%@ taglib uri="http://jakarta.apache.org/taglibs/dbtags" prefix="sql" %>
```

```
<%-- open a database connection --%>

<sql:connection id="conn1">
  <sql:url>jdbc:mysql://localhost/test</sql:url>
  <sql:driver>org.gjt.mm.mysql.Driver</sql:driver>
</sql:connection>

<%-- insert a row into the database --%>

<sql:statement id="stmt1" conn="conn1">
  <%-- set the SQL query --%>
  <sql:query>
      insert counters(page,hitCount) values('<%=request.getRequestURI()%>', 0)
  </sql:query>
  <%-- the insert may fail, but the page will continue --%>
  <sql:execute ignoreErrors="true"/>
</sql:statement>

<%-- update the hit counter --%>

<sql:statement id="stmt1" conn="conn1">
  <%-- set the SQL query --%>
  <sql:query>
      update counters set hitCount = hitCount + 1 where page like '<%=request.getRequestURI()%>'
  </sql:query>
  <%-- execute the query --%>
  <sql:execute/>
</sql:statement>

<P>This page has been hit

<%-- query the hit counter --%>

<sql:statement id="stmt1" conn="conn1">
  <sql:query>
    select hitCount from counters where page like '<%=request.getRequestURI()%>'
  </sql:query>
  <%-- process only the first row of the query --%>
  <sql:resultSet id="rset2" loop="false">
      <sql:getColumn position="1"/>
  </sql:resultSet>
</sql:statement>

times. The page counter is implemented using the Jakarta Project's
DBTags tag library, calling JDBC indirectly.</P>

<%-- close a database connection --%>
<sql:closeConnection conn="conn1"/>

</BODY>
</HTML>
```

皆さんはどうか存知ませんが、私はいささかがっかりしています。これではスクリプトレットの例よりもかえってわかりにくくなったように思え、これに満足しそうなプログラミングを行わないHTMLページ設計者はいるのでしょうか。しかし、どこを間違ったのでしょうか? あれほど、人々のアドバイスに従って、スクリプトレットを取り除き、カスタム・タグに置き換えたのに。

> カスタム・タグ・ライブラリーの開発は比較的簡単ですが、ある程度の熟考が必要で、時間のかかる作業です。私はよく、タグ・ライブラリーの作成者は最初にスクリプトレットを使ってタグ動作をプロトタイプ化し、それからそのスクリプトレットをタグに変換するようにアドバイスしています。

代替案としては、Allaire's JRun Server Tags (JST) を使用することがあげられますが、これは各タグをJSPページ (.jst 拡張子がある)としてオーサリングすることでタグ・ライブラリーをプロトタイプ化できるようにしてくれます。JSTは実行時にそのページをタグ・ハンドラーに変換するので、JSTテクノロジーがクライアント・ページに透過的になります。Allaireは「J2EEコミュニティーのあらゆるメンバーがそのメリットを活用できるようにポータブル・テクノロジーとしてJSTを確立することが目標だ」と主張していますが、JSTは現在JRunでしか提供されていません。JSTがタグ開発の手段としてさらに普及するかどうかは、時がたてば明らかになるでしょう。一方、我々はスクリプトレットがタグのビジネス・ロジックを開発するための優れた基礎をもたらしてくれると考えます。ロジックがデバッグされた後、これをタグ・ハンドラー・クラスに移行します。

タグ・ライブラリーについて人が教えてくれないのは、タグ設計が言語設計であるということです。現在までのところ、作成されているほとんどのタグ・ライブラリーは、プログラマーのためにプログラマーによって作成されていました。こうしたタグのセマンティクスは、他のプログラマーに合わせて調整されます。さらに、モデルとプレゼンテーションの分離はご記憶にあるでしょうか? これはDBTagでは十分にサポートされません。 `sql:getColumn` タグは、`jsp:getProperty` アクションと類似しています。つまり、タグの結果を直接出力ストリームに送り出します。そのため、出力を望ましいフォームにもたらすこととDBTagを使うことを区別することが困難になります。最後に、リスト3とリスト4の間のロジックの相違点に注意してください。DBTagの `execute` タグは、JDBCを介して送られた `update` ステートメントの更新カウントを破壊します。照会結果だけがリトリーブできます。これはつまり、 `update` ステートメント によって更新された行数を突き止めることはできないということです。ですから、 `update` ステートメントと `insert` ステートメントを切り替える必要があります。我々は常に、新規レコードを挿入し、DBTagsがエラーを無視するよう強制し、それから更新を実行するよう試みます。

DBTagsタグ・ライブラリーに公正を期して言うならば、これはプログラマーにとって好ましくないタグではありません。更新カウントを破壊することを別にすれば、コードはかなり良好なJDBCへのマッピングを行います。ただし、そこに問題が潜んでいます。タグがもたらすのは、JDBCパッケージの直接変換と大差ないものなのです。一部の例外処理を隠す以外に、タグ・ライブラリーはスクリプトレットに対し実際には何ら抽象をもたらしてはいません。もちろんこれは、プレゼンテーションを機能と分離する上では役に立ちません。

ですから真の問題は、スクリプトレットとタグのどちらを使用するのかということではないのです。それ自体は、プレゼンテーションから機能を分離するという問題の原因ではなく、結果なのです。その解決策は、適切な論議のレベルでプレゼンテーション・ページ作成者に対してより高レベルの機能性を提供することです。タグがスクリプトレットよりもよいと考えられている理由は、スクリプトレットがその名のとおりプログラミングを行うのに対し、タグは高レベル概念を表すことができるということです。

プレゼンテーション・ページからJDBCを隠す

JDBCをJSPテクノロジーに統合する場合は、できる限りプレゼンテーションの作成者から統合を見えないようにしたいと考えます。データベース概念を公開する場合は、適切な抽象化レベルで公開したいと考えます。このアプローチは、次の例を導きます。

リスト5の例では、JDBC統合をプレゼンテーション・ページから隠しています。

リスト5. 隠されたJDBCのあるJSPページ

```
<jsp:directive.include file="/pagelets/hitCounter.jsp" />
<HTML>
<HEAD>
<TITLE>JDBC hidden example</TITLE>
</HEAD>
<BODY>
<P>This page has been hit
<jsp:getProperty name="hitCounter" property="int" />
times. The page counter is implemented indirectly: a JavaBeans component containing the
hit count is inserted into the environment and referenced within the page using
the JSP getProperty action. The JSP page doesn't have any exposure to JDBC.</P>
</BODY>
</HTML>
```

インクルードされている `hitCounter.jsp` ファイルは、環境設定を管理します。コンテンツは、スクリプトレット、タグ、あるいは単なる `taglib` ディレクティブでかまいません。コンテンツは、プレゼンテーション・ページにとって望ましい環境を確立するものであればどのようなものでも可能です。希望があれば、 `getProperty` アクションをカスタム・タグと置き換えることもできます。たとえば、次のようにします。

```
This page has been hit
<page:hitcounter />
times.
```

先ほど述べたように、これらのヒット・カウンターの例は純粋に説明のために紹介したものです。各ページにこのようなデータベース・オペレーションを行うことは、不要な資源の使用を招くことになります。上記の例は、実際にヒット・カウンターをどのように表示させるかを示しています。カスタム・タグでこれを隠すことにより、この実装を完全に隠しています。これで、ヒット・カウント情報を実行時に総合して、データベースを定期的 (たとえば、各セッションの終了時) に更新できます。保管の手段(データベースなど) でさえプレゼンテーション・ページの作成者から隠されます。これは、DevTechで我々がヒット・カウンターを実装する方法です。 `bean` クラスにヒット・カウンター・モデルの振る舞いを実装させています。タグはその振る舞いを我々のページに取り込みます。

JavaBeansコンポーネントとの統合

これまでの例はかなり単純なものでしたが、ほとんどのデータベース・オペレーションは、こうした簡単な照会と更新に比べてはるかに高度なものになります。これまでのところ、JSPページでJDBCを使う場合の基本原則をいくつか扱ってきましたが、少し複雑で、もちろんより一般的なアプリケーションのタイプをあげて話を結ぶことにしましょう。

このセクションの例 (以下の リスト9) は、Webサイトのビジターが書き込んで内容を充実させていくコンテンツを、サポートする1つの方法を示しています。つまり、ビジターがURIに関連づけられたデータベース・コンテンツを読み、寄稿することでコンテンツを増やすようにしたいと考えています。このようなコンテンツは、最新のWebサイトではかなり一般的になっています。同様の基本部分を利用して、次のものを構築できます。

- Amazon.comで目にするようなレビュー・ページ
- リンク・ページ
- 掲示板

- Wikiweb

この例で紹介する若干手の込んだJSPコンポーネントのバージョンにより、多彩な技術的経歴を持つ複数の設計者が作成したような非常に外観の異なる複数のWebページを実装できます。これらのページに共通していると思われる唯一の点は、ビジター寄稿のコンテンツに対応することです。

我々の注釈の例では、HTMLフォームを使用します。HTMLフォームをJSPと共に使用する場合は、プロパティーがフォーム・フィールドに対応付けられるBeanを使うのが便利です。そうすることで、 `setProperty` タグが独自のマジックを見せてくれます。

リスト6. フォームに対応するBeanインスタンス

```
<%-- setup a bean instance that matches our form --%>
<jsp:useBean id="instance-name" class="bean-class" ... />
<%-- set all bean properties that match a form field --%>
<jsp:setProperty name="instance-name" property="*" />
```

BeansとResultSetsのマッピング

例では、説明を簡潔にするために `com.devtech.sql` を使用しています。Javaリフレクションおよび イントロスペクションが、JDBCデータとBeanプロパティーの間のマッピングを提供する列およびプロパティー名に対して使用されています。DevTechパッケージを独自のコードに置き換えてかまいません。

JavaBeansコンポーネントとの統合は、JSPテクノロジーの設計における優れた側面の1つです。残念なことに、BeansとJDBCの間の統合はとてもシームレスとはいえないので、DevTechでのJDBCの取り組みには、BeansとJDBCの間の統合だけでなく、必要な例外処理も備えるパッケージを開発して、プログラマーが詳細事項に対処しなくてもすむようにしました。

この注釈の例では、 `com.devtech.sql` パッケージから2つの照会および更新のメソッドを用いています。ここで使用されている照会メソッドでは、 `bean` クラス、SQL照会、照会のプレースホルダーを入れる `Object` アレイを渡します。この例の場合、ページのURLに対するものが唯一のプレースホルダーになります。結果は、基本的にはイテレーターのタイプであるデータベース・カーソル・オブジェクトです。

リスト7. データベース・カーソル・オブジェクト

```
dataBase.queryCursor(AnnotationDBBean.class, new String[] { URL },
 "select page, author, annotation, DATE_FORMAT(whenPosted, '%W %d%b%y %T')" +
 " as whenPosted from annotations where page like ?");
```

この照会メソッドを興味深いものにしているのは、指定されているタイプのBeanがインスタンス化され、名前が `ResultSet` の列名と一致するBeanプロパティーが自動的にその値を設定するという点です。カーソルを使って次の行を選択するたびに、Beanのプロパティーが `ResultSet` から自動的に設定されます。

ここで使用される更新メソッドでは、 `bean` インスタンス、 `String` アレイ、 `update` ステートメントを取り込みます。 `String` アレイの値は、更新のプレースホルダーを埋めるために使用される望みのBeanプロパティーを指定します。この場合、 `page` 、 `author` 、 `annotation` プロパティーがBeanから選択されています。

リスト8. 更新メソッド

```
int count = dataBase.update(annotationBean,
 new String[] { "page", "author", "annotation" },
 "insert into annotations(page, author, annotation) values(?, ?, ?)");
```

例にあげたJSPページannotations.jsp, を リスト9 に示します。ハイライト表示されているセクションは、リスト10に示すようにカスタム・タグと置き換えることもできるスクリプトレットをいくつか示しています。ページの残りの部分は、ページ設計者を補助するために用意されているJSPコメント、動的コンテンツをページに配置する `getProperty` アクション、および標準HTMLで構成されています。JSPコメントを使用しているのは、それらがプライベートであり、出力ストリームに表示されないためです。

リスト9. 注釈用のJSPページ

```
<jsp:directive.include file="/pagelets/annotate.jsp" />

<%--
By the time we arrive here, the annotation bean has been established, and if the
form is submitted, the contents will be posted to the database. The page
property is initialized. If the author is known during this session, that property
is also initialized.
Bean:           "annotation"
Properties:     String page;
                String author;
                String annotation;
                String whenPosted;
Access to any bean property follows the format:
        <jsp:getProperty name="annotation" property="property-name" />
--%>
<HTML>
<HEAD>
<TITLE>Comments for <jsp:getProperty name="annotation" property="page" /></TITLE>
</HEAD>
<BODY>
<p align="left"><font size="+1">
Comments for <i><jsp:getProperty name="annotation" property="page" /></i>
</font>.</p>
<CENTER><HR WIDTH="100%"></CENTER>
<!-- Annotation Submission Form -->
<FORM method="POST">
<TABLE>
<TR>
        <TH align="left">Name:</TH>
        <TD><INPUT type=text name=author size=50 maxlength=60
        value="<jsp:getProperty name="annotation" property="author" />"> </TD>
</TR>
<TR>
        <TH valign="top" align="left">Note:</TH>
        <TD><TEXTAREA name=annotation cols=40 rows=5 wrap=virtual>
        <jsp:getProperty name="annotation" property="annotation" /></TEXTAREA></TD>
</TR>
<TR>
        <TD align="center" colspan="2"><INPUT type=submit value="Add Comment"></TD>
</TR>
</TABLE>
</FORM>
<!-- End of Annotation Submission Form -->
<!-- beginning of annotations -->
<%--
The following section iterates through all annotations in the database for the
requested page. To change the look of the page, just change anything in the
demarcated area.
```

```
--%>
<jsp:scriptlet>
Database.Cursor annotations = annotation.getCursor();
while (annotations.next(annotation) != null)
{
</jsp:scriptlet>
<%-- beginning of annotation change area --%>
<CENTER><HR WIDTH="100%"></CENTER>
From: <jsp:getProperty name="annotation" property="author" /></A>
at <jsp:getProperty name="annotation" property="whenPosted" /><BR>
<jsp:getProperty name="annotation" property="annotation" /><BR>
<%-- end of annotation change area --%>
<jsp:scriptlet>
}
annotations.close();
</jsp:scriptlet>
<!-- end of annotations -->
</BODY>
</HTML>
```

対応するカスタム・タグは明確ですが、あまり情報価値がありません。

リスト10. 対応するカスタム・タグ

```
<sql:results queryName="annotations" bean="annotation">
<CENTER><HR WIDTH="100%"></CENTER>
From: <jsp:getProperty name="annotation" property="author" /></A>
at <jsp:getProperty name="annotation" property="whenPosted" /><BR>
<jsp:getProperty name="annotation" property="annotation" /><BR>
</sql:results>
```

この例では、プログラマーである皆さんに何が行われているかを示すためにスクリプトレットを使用しているだけです。これが宣言タグに置き換えられたとすれば、ページ設計者にとっては明確になりますが、皆さんにとっては情報価値のないものになってしまいます。

ロジックは単純明快です。 `annotation.getCursor()` 呼び出しは、サーバーへの接続を獲得し、照会を発行し、結果セットにデータベース・カーソル・オブジェクト `annotations` をセットアップします。 `annotations.next()` が呼び出されるたびに、新しい行が結果セットから取り出され、その値はBeanに移され、そのBeanへの参照がメソッドから返されます。ここで使用されている特定の `next()` メソッドは、値を入れるべきBeanのパラメーターを取ります。カーソルに、行ごとに新しいBeanをインスタンス化させることもできますが、Beanを再利用する方が効率的です。

実際の照会と更新のいずれもプレゼンテーション・ページには提示されないことに留意してください。プレゼンテーション・ページの環境をセットアップする組み込みのページにも、`setProperty` および `update` アクションが含まれています。これらのアクションは、プレゼンテーション・ページから独立しています。 `annotation` Beanのプロパティーによって具体化されているコントラクトだけが重要です。これは、プレゼンテーションをModelの振る舞いから分離するポリシーと調和しています。ページ設計者は、プレゼンテーションが行われる方法を全面的に変更できますが、データベースが統合される方法には手を加えることができません。データベースの更新または照会において変更が行われるとすれば、それはJSPプログラマーにゆだねられます。

要約

これで、JavaServer Pages、JavaBeans、JDBCテクノロジーを組み合わせ、リレーショナル・データベースを通じて動的コンテンツを生成する方法の紹介を終わります。この記事では、新参のJSPプ

ログラマーにとって最も明白なアプローチであるスクリプトレットから取りかかりました。スクリプトレットの制限されない使用により、ロジックとプレゼンテーションが絡み合ってしまい、いずれも維持することが困難になってしまう様子を目の当たりにしました。さらに、タグ・ライブラリーが必ずしもMVCの分離を改善するわけではないこと、そしてタグがプログラミング用語で表現されても、これらを利用するページがページ設計者にとって分かりやすいものにはならないことを思い知らされました。最終的に、我々は、コンテンツのプレゼンテーションとデータベース・アクセスを切り離すいくつかの方法を示すもっと複雑な例を検討しました。

これで皆さんは、実際のデータベース・アクセスをページ設計者に見せずに、データベース・コンテンツをWebサイトに統合する方法について、基本的な概念を把握できたはずです。また、プログラマーとしての皆さんにとって一番参考にならない例が、ページ設計者に最も適した例だということにも着目してください。JSPソリューションを計画するにあたっては、ページ設計者を念頭に置くようにしてください。

関連トピック

- JavaServer Pages テクノロジーの公式ホーム・ページは、JSPテクノロジーに関するあらゆる公式ドキュメントを含む膨大な情報に触れることができる絶好の開始点です。
- Jakarta Project の公式ホーム・ページには、Java ServletおよびJavaServer Pagesテクノロジー用の参照実装例が掲載されており、他の多くの関連オープンソース・アクティビティーのハブとしての役割を果たします。
- IBM developerWorks Javaテクノロジー・ゾーン では、JavaServer Pagesテクノロジーを含むJavaのすべてに関するベンダー中立のチュートリアルと記事を提供しています。
- Greg Travis著の 「An easy JDBC wrapper」 (developerWorks、2001年8月) は、基本データベース使用を簡単にするシンプルなラッパー・ライブラリーについて説明しています。
- 「What's new in JDBC 3.0?」 (developerWorks、2001年7月) は、Java Database Connectivity 3.0の仕様に備わった新たな特色と機能強化の概要を説明しています。
- JDBC の公式ホーム・ページでは、役立つ参考文献をそろえています。