DAO 生成ツール使用ガイド(DB 編)

概要

当資料は、DAO生成ツールの機能、および、DAO生成ツールによって生成されるクラスの機能に関するガイドです。

- 目的

DAO生成ツールの機能および使用方法の理解

- 要約

DAO生成ツールとは、J2EEのDAO・DTOパターンを構成するクラス群をDDL/SQLから自動生成するツールです。Rational Application Developer(RAD)のプラグインとして動作します。DAO・DTOクラス群を生成する手順は次のとおりです。

「プロジェクト基本情報の作成」→「DTOの設定」→「DAOの設定」→「コード生成」

ガイドでは各手順の詳細について説明します。また、当ガイドでは生成したクラス群の使用方法についても説明します。

- 適用範囲

DAO生成ツールを使用するプロジェクト全般

- 前提事項

当ガイドは DAO 生成ツールが既にインストールされていることを前提としています。インストール方法の詳細は「Web application components インストール・ガイド」をご覧ください。

- 対象読者

DAO生成ツールおよびその生成クラスを使用する開発者全般

- 前提スキル

J2EE、SQL、Eclipse、および、O/Rマッピングの概要を理解していることが望ましい

- 商標

変更履歴

改訂番号	改訂日	主要な変更内容	更新者
1.0	2005/04/08	初版	IBM
1.1	2005/11/17	バージョンアップ(ver1.0→ver1.1)に伴う差分の更新	IBM
1.2	2006/02/17	バージョンアップ(ver1.1→ver1.2)に伴う差分の更新	IBM
1.3	2006/04/03	バージョンアップ(ver1.2→ver1.3)に伴う差分の更新	IBM
1.3.1	2006/06/29	バージョンアップ(ver1.2→ver1.3)に伴う差分の更新、構成の見直し、図表の追加など	IBM
1.4	2007/03/09	バージョンアップ(ver1.3→ver1.4)に伴う差分の更新	IBM
1.4.0.2		カーソル更新・カーソル削除の編集・削除についての注意点を追記 プロパティファイルの記述時の注意点を追記 CREATE文、ALTER文の対応一覧を追記	IBM
1.4.0.3	2007/04/27	SQL文に関する注意点を追記	IBM
1.4.0.4	2007/07/27	使用可能なJavaデータ型と列データ型の対応一覧を追記 使用可能なDDL文一覧を更新 誤植の修正	IBM
1.5	2007/10/12	バージョンアップ(ver1.4→ver1.5)に伴う差分の更新、図表の更新、DTOクラスからDAO定義情報を作成する機能説明の追加など	IBM
1.5.1	2007/12/10	バージョンアップ(ver1.5→ver1.5.1)に伴う差分(動的パラメーター照会機能)の更新	IBM
2.0	2008/04/25	バージョンアップ(ver1.5.1→ver2.0)に伴う差分の更新(Spring連携機能など)	IBM
2.0.1	2008/07/11	バージョンアップ(ver2.0→ver2.0.1)に伴う差分の更新(バッチ更新機能など)	IBM
2.0.2	2008/08/19	次の内容を追記 ・Parameterクラス使用時の注意点を追記 ・誤植の修正	IBM
2.1	2008/10/24	次の内容を追記 ・RAD V7.5対応に伴う図表更新 ・新機能の説明を追加	IBM
2.2	2008/12/05	バージョンアップ(ver2.1→ver2.2)に伴う移行手順の更新、および、Explain結果整形用Antタスクの説明を追加	IBM
7.0	2009/02/20	コピーライト表記の修正、ツール名、用語の再定義	IBM
7.0.2	2009/09/25	読み込み可能な VIEW の書式を追記	IBM
7.0.3	2009/12/16	DB 種別に関する注意点を追記	IBM
7.0.4	2010/03/26	SQL 単体読み込み機能、データソース指定機能の追加に伴う画面の更新	IBM
7.0.5	2010/10/29	次の内容を追記 ・動的 IN 句に関する記述 ・DDL 読み込み時のオプション追加に関する記述 ・Ant タスクに関する記述 ・Logger 実装 ・DefaultLogger によるログ出力例とマスキング機能の出力例を追加 ・誤植の修正	IBM
7.0.6	2011/04/15	次の内容を追記 ・ DAO 生成ツールの特長を追加 以下の章を追加 7.2Excel設計書との連携 次の内容を修正 ・表 4-5 照会系メソッド一覧の誤植を修正	IBM

		・表の行の途中の改ページを修正 ・表の改ページ時の見出しを修正 SQL 定義ファイルの改行出力対応に伴い以下を修正 ・4.2.2使用可能なSQLの修正 ・6.2.3 ExceptionHandlerクラス の修正 ・7.5バージョン間の差異についてについて への追記	
7.1.0	2011/07/29	次の内容を追記 ・ SQLJ 実装での WHERE CURRENT OF 句の誤植を修正 ・ SQLJ 実装での SELECT INTO への変換条件を追記	IBM
7.1.1	2012/1/27	次の内容を修正 ・ 図 3-3『DTO定義』エディターおよび図 3-5『DBテーブル項目追加』ダイアログのイメージ差し替え ・ 表 3-3 「DBテーブル項目」属性一覧に結合列の項目の説明を追加 ・ 表 4-5 照会系メソッド一覧および表 4-6 更新系メソッド一覧に脚注、表 4-7 メソッドのパターン一覧に可変個のprimitiveラッパー型引数の説明を追加 ・ 表 5-1 プロパティー一覧 に「 retrieveArgument 」「 argumentType 」「 trim 」「optimizeForSelectIntoStatement」の項目を追加 ・ 表 7-7 DAO設計書のメソッド仕様シートの項目のメソッド名の記述を修正 ・ 7.2.8 Excelインポートにおける注意事項の記述を追加 ・ 7.5 バージョン間の差異についての記述を追加 ・ SQLJ 実装で単一行照会を使用する場合の制約を追記	IBM
7.1.1.1	2012/03/05	・ 表 5-1 プロパティー一覧に「accessorNameStrict」の項目を追加 ・ 7.5 バージョン間の差異についての記述を追記 ・ primitive ラッパー型引数の制約を追記	IBM
7.3.0	2012/11/30	・ 前提事項から製品のバージョン表記を削除 ・ 誤植修正	IBM

目次

1. 概要 6
1.1. DAO 生成ツールとは 6
1.2. DB アクセス機能とは 6
1.3. DAO 生成ツールの特長 8
1.3.1. DAO 実装形態の選択 8
1.3.2. 継承を利用したコードのカスタマイズ 8
1.4. DAO 生成ツールによるコード生成の流れ 9
2. プロジェクト基本情報の設定 10
2.1. プロジェクト基本情報の新規作成 10
2.2. プロジェクト基本情報の編集と操作 12
3. DTO の設定 13
3.1. DTO 定義情報の新規作成 13
3.2. 読み込み可能な DDL 15
3.2.1. CREATE TABLE 15
3.2.2. ALTER TABLE 15
3.2.3. CREATE VIEW 16
3.2.4. 使用可能な Java データ型と列データ型 16
3.3. DTO 定義情報の編集と操作 16
3.3.1. DB テーブル項目 18
3.3.2. 関連項目 19
4. DAO の設定 21
4.1. DAO 定義情報の新規作成 21
4.2. DAO 定義情報の編集と操作 22
4.2.1. ユーザー定義 CRUD メソッドの追加 26
4.2.2. 使用可能な SQL 29
4.2.3. ユーザー定義 CRUD メソッドの編集 32
5. コード生成 41
5.1.1. コードの生成 41
5.1.2. SQL 定義ファイルについて 45
6. 生成クラスの使用方法 46
6.1. 生成クラス概要 46
6.2. API の詳細 48
6.2.1. DAO クラス 48
6.2.2. Parameter クラス 52
6.2.3. ExceptionHandler クラス 53
6.2.4. Logger クラス 54
7. その他の情報 57
7.1. Spring との連携 57
7.1.1. Spring とは 57
7.1.2. 事前準備 57
7.1.3. 利用方法 59
7.2. Excel 設計書との連携 60
7.2.1. Excel 設計書との連携に使用する Excel ファイルの取得方法 60
7.2.2. Excel 設計書による DAO 生成の流れ 60
7.2.3. DTO 設計書テンプレート 61
7.2.4. DAO 設計書テンプレート 62
7.2.5. DTO 設計書生成ツール 64
7.2.6. DAO 生成ツール上での Excel ファイルのインポート 66
7.2.7. DTO/DAO 設計書テンプレートのカスタマイズ 68
7.2.8. Excel インポートにおける注意事項 69
7.3. Ant 69
7.3.1. DDL 読み込み 69
7.3.2. SQL 読み込み 70

7.3.3. コード生成 71
7.3.4. 実行可能 SQL の生成 71
7.3.5. Explain 結果のレポート 72
7.3.6. Explain 結果の比較レポート 73
7.4. 制約事項 75
7.5. バージョン間の差異について 75

1. 概要

この章では、DAO 生成ツールの概要と DAO 生成ツールによるコード生成作業の流れを説明します。

1.1. DAO生成ツールとは

DAO 生成ツールとは、DAO(Data Access Object)/DTO(Data Transfer Object)を自動生成するツールです。J2EE の DAO パターンに分類されるクラス群、および、DTO パターンに分類されるクラスなどを生成します。DAO 生成ツールを使用することにより、業務ロジックとアクセス・ロジック（永続層との仲介を行なう層）の分離が可能となります。
DAO 生成ツールでは、データの保管先である外部リソースに、DBMS あるいはテキストファイルのいずれかを選択することが可能です。当ガイドでは、外部リソースにDBMSを選択した場合に使用する、「DBアクセス機能」について説明します。
なお、DAO 生成ツールは Rational Application Developer for WebSphere Software (以降、RAD)のプラグインとして動作します。

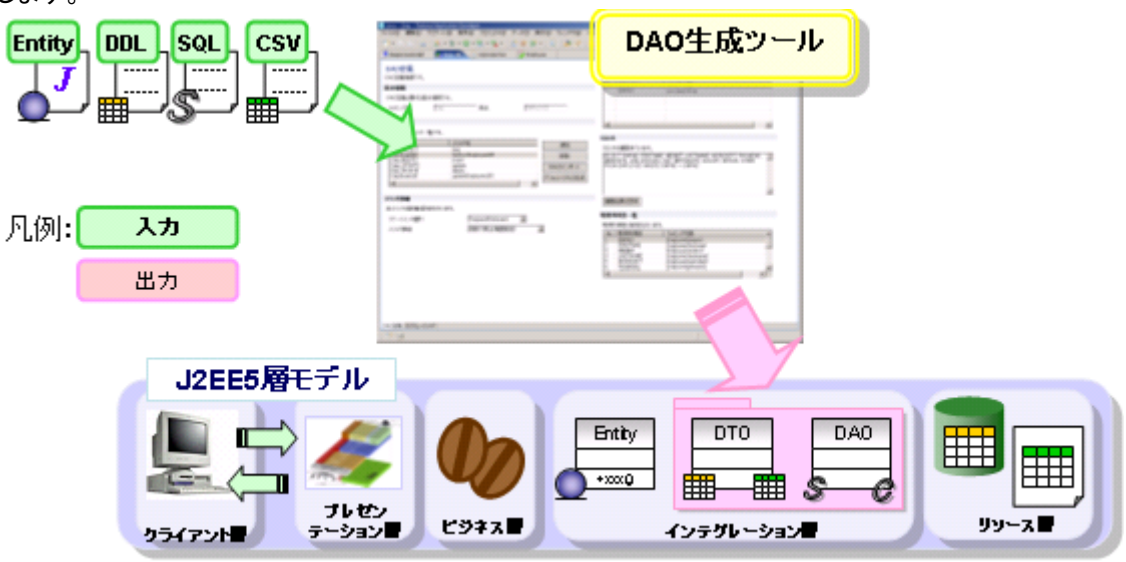


図 1-1DAO 生成ツール概要

1.2. DBアクセス機能とは

DAO 生成ツールの DB アクセス機能とは、O/R マッピング(Object Relational Mapping)の考え方に基づき、DDL や SQL から DB アクセス・コンポーネントを自動生成する機能です。具体的には、J2EE の DAO(Data Access Object)パターンに分類されるクラス群、および、DTO(Data Transfer Object)パターンに分類されるクラスなどを生成します。
DAO 生成ツールを使いこなすことにより、業務ロジックと DB アクセス・ロジックの分離や SQL の一元管理が可能となります。

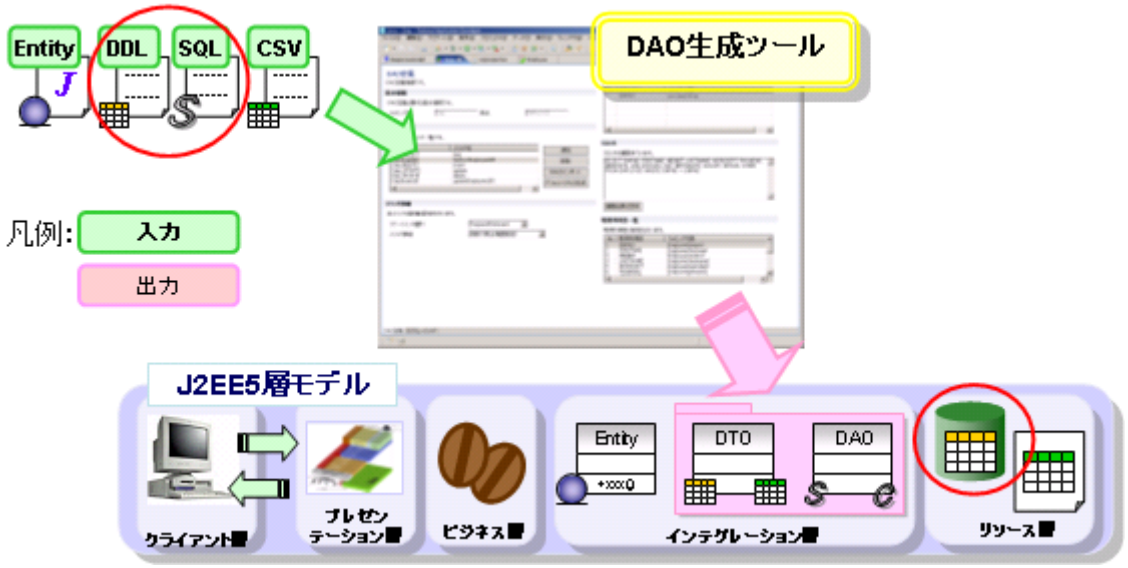


図 1-2 DAO 生成ツールの概要

(参考) DAO 生成ツールでは、DAO・DTO の役割をそれぞれ次のように定義しています。
DAO：SQL の発行および結果表の取得、DTO の生成、DTO からのパラメーター取得
DTO：結果表のレコードの保持、更新時のパラメーター保持

DAOとDTOの関係は次図のとおりです。DAOのメソッドが持つ各機能の詳細については、後述の「6生成クラスの使用方法」をご覧ください。

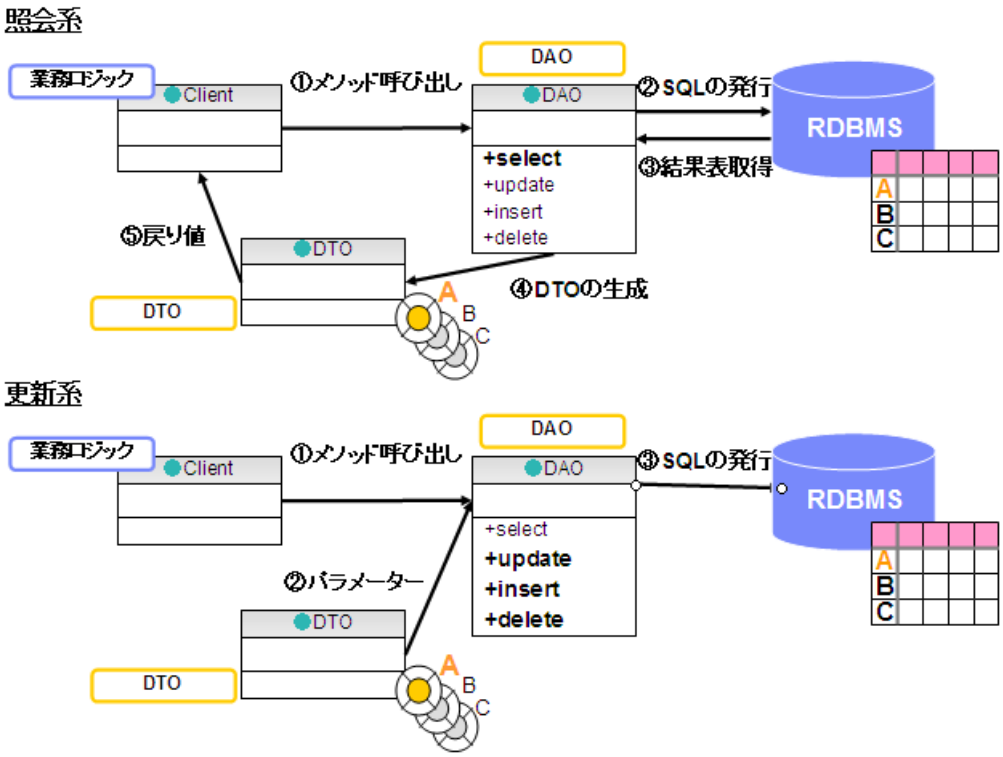


図 1-3 DAO と DTO の関係

1.3. DAO生成ツールの特長

1.3.1. DAO実装形態の選択

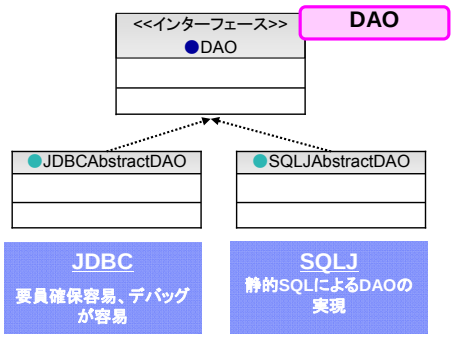


図 1-4 実装形態の選択

DAO生成ツールのDBアクセス機能では、DAOクラスの生成時にその実装形態として「JDBC」または「SQLJ」のいずれかを選択することが可能です(Express版の場合は、Spring Frameworkを前提とした両実装形態のいずれかを選択します)。ここでSQLJとは、Java言語上で「静的SQL」を使用するためのANSI仕様です。アクセス・パスの固定やコンパイルによるSQLエラーの検出、SELECT INTO句による高速照会などの特徴があります。次表にJDBCとSQLJの対比を示します(http://www.ibm.com/developerworks/jp/data/library/ds/techdoc/javasqlj.htmlより引用)。選択時の参考資料としてご利用ください。

表 1-1 JDBC と SQLJ の対比(下記リンクより引用)

	JDBC	SQLJ
規格	Sun (part of J2EE)	ISO/ANSI（J2EE の一部ではない）
SQL 仕様レベル	なし（最低限、Entry Level SQL-92 のサポートが必要）	SQL-1999
セキュリティー	平均	強力
パフォーマンス	低速（アプリケーション・プログラム実行時に動的アクセス・プランを作成）	高速（開発時に静的アクセス・プランを作成
構文	低（煩雑）	高（簡潔）
SQLJ と JDBC の相互運用性	なし	あり
型チェックとスキーマ・チェック	緩やか（ランタイム時に実行）	厳密（開発時に実行）

1.3.2. 継承を利用したコードのカスタマイズ

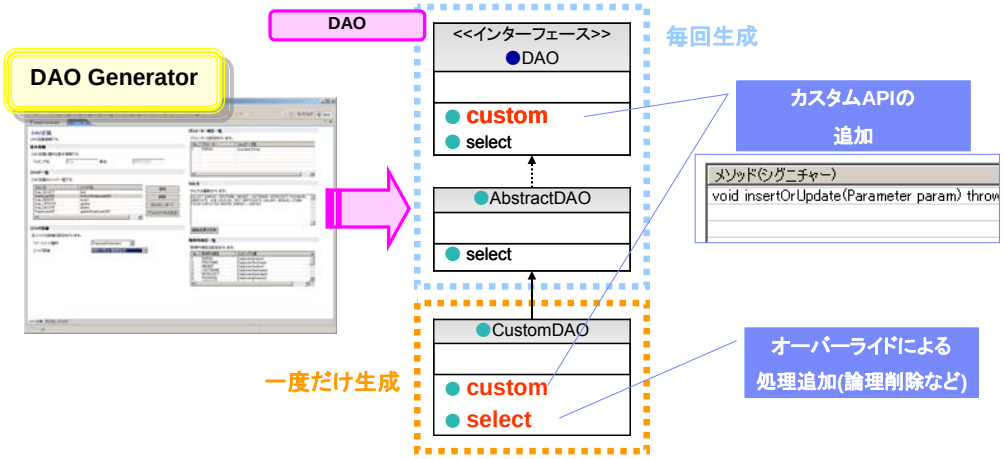


図 1-5 継承を利用したコードのカスタマイズ

DAO Generator では、生成された CRUD 操作のカスタマイズが可能です。新たな CRUD 操作を追加する場合は、『追加メソッド』にカスタム API を追加することにより、コード生成の際、インターフェースにメソッドが追加されます。自動生成されたコードを編集する場合は、CustomDAO を編集します。CustomDAO は、一度しか生成されないため、コードを編集しても上書きされることはありません。

1.4. DAO生成ツールによるコード生成の流れ

DAO 生成ツールによるコード生成の流れは次のとおりです。まず DB アクセス・コンポーネントの管理単位である「プロジェクト」の基本情報を設定し、次に、生成する DAO・DTO の設定を行います。最後に、設定した情報を元に DAO・DTO クラスを生成します。DAO・DTO の設定作業は、生成する DAO・DTO クラスの数(通常は使用する表の数)だけ繰り返す必要があります。なお、図中の番号は当資料の章番号に対応しています。

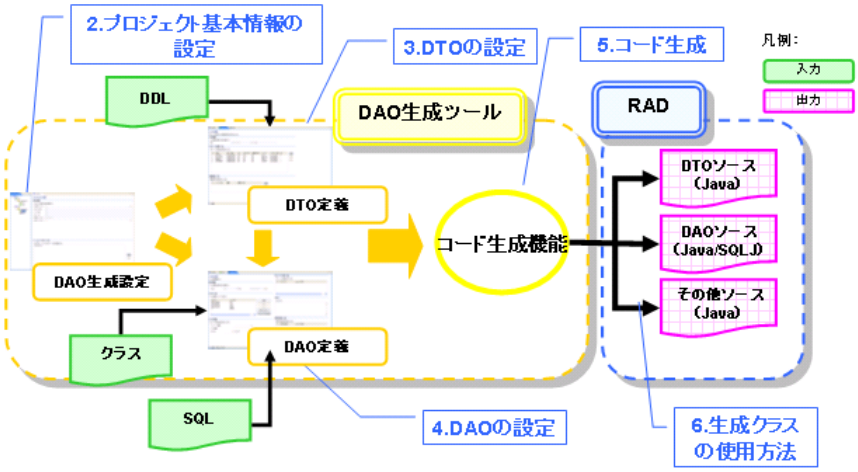


図 1-6 コード生成の流れ

以降の章では、コード生成作業手順に沿って、DAO 生成ツール自体の機能(2～5 章)、および、生成されたコードの機能(6 章)を説明していきます。各章の概要は次のとおりです。

- 2 プロジェクト基本情報の設定
  - DAO 生成ツールの DB アクセス・コンポーネント管理単位である「DAO 生成設定」の作成方法や設定方法について説明。
- 3 DTOの設定
  - DAO 生成ツールが生成する DTO クラスの設定方法、表に含まれる列と DTO が持つフィールドの関係、および、表の関連と DTO 参照の関係などを説明。
- 4 DAOの設定
  - DAO 生成ツールが生成する DAO クラスの設定方法、SQL と DAO が持つメソッドの関係、および、SQL の取得列・パラメーターと DTO のフィールドの関係などを説明。
- 5 コード生成
  - 実装コードの生成方法を説明
- 6 生成クラスの使用生成クラスの使用
  - 生成されたコードの概要、および、各 DAO が持つメソッドの使用方法を説明。
- 7 その他の情報
  - 制約事項、バージョン間の差異などについて説明。

2. プロジェクト基本情報の設定

DAO 生成ツールは、DB アクセス・コンポーネントを「プロジェクト」という単位で管理します。この章では、プロジェクトの作成方法・設定方法について説明します。

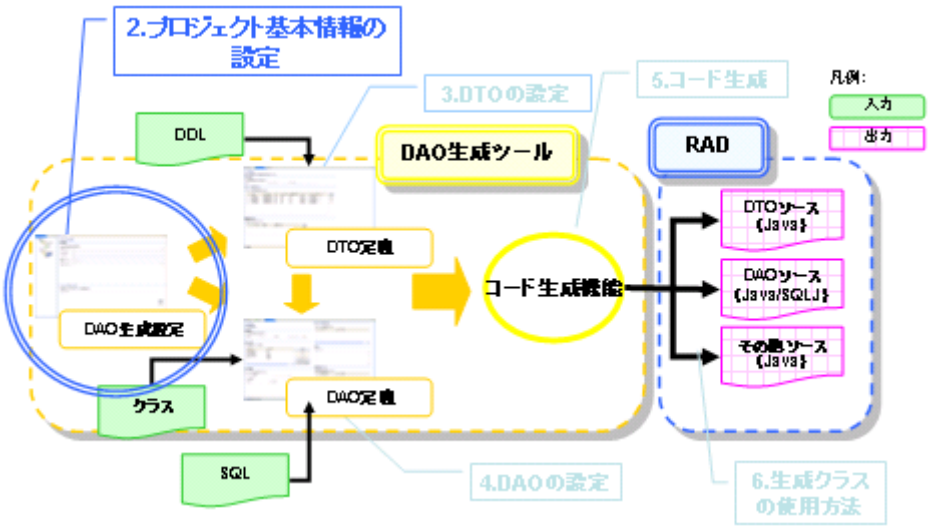


図 2-1 プロジェクト基本情報の設定

2.1. プロジェクト基本情報の新規作成

DAO 生成ツールでは、まず「DAO 生成設定」を新規作成し、プロジェクトの基本情報を入力する必要があります。基本情報には DAO・DTO 定義情報の格納先、および、生成コードの出力先などの情報が含まれます。「DAO 生成設定」の新規作成手順は次のとおりです。

1. 『新規 DAO 生成設定ファイル作成』ウィザードの起動
2. プロジェクト基本情報の入力

『新規 DAO 生成設定ファイル作成』ウィザードは、RAD のメニューバーから『ファイル』-『新規』-『その他』を選択し、さらに、ウィザード選択ダイアログから『Web application components』-『DAO 生成設定ファイル』を選択して『次へ』ボタンを押下すると起動します。

次に、『新規 DAO 生成設定ファイル作成』ウィザードに含まれる、各画面項目とイベントについて説明します。このウィザードでは、プロジェクトに共通な各種基本情報を入力してください。画面項目の中には、DAO 生成設定ファイル作成後に変更ができないものも含まれているので、注意してください。

図 2-2『新規 DAO 生成設定ファイル作成』ウィザード

No.	項目名	書式	説明
1	永続化ターゲット	プルダウン	DB アクセス機能を使用する場合は、「DB」を選択。作成後、変更不可。
2	プロジェクト	プルダウン	RAD 上のプロジェクト名を選択。各種設定情報や生成コードの出力先となる。作成後、変更不可。
3	ファイル名	全角文字	任意の文字列を入力。作成されるプロジェクト設定の名前。実際には「入力された文字列」+「.dgdx」というファイル名になる。
4	プロジェクト名	全角文字	任意の文字列を入力。作成後、変更不可。入力した値は全てのソースコードのヘッダーに含まれる。
5	システム名	全角文字	任意の文字列を入力。作成後、変更不可。入力した値は全てのソースコードのヘッダーに含まれる。
6	コピーライト	全角文字	任意の文字列を入力。作成後、変更不可。入力した値は全てのソースコードのヘッダーに含まれる。
7	基準ディレクトリー	半角英数字	各種設定情報の格納先となるディレクトリー名を指定。作成後、変更不可。
8	出力ディレクトリー	プルダウン	生成コードの出力先ディレクトリー名を選択。作成後、変更不可。
9	ソースコード・パッケージ名	半角英数字	生成コードの基準となるパッケージ名。

表 2-1 画面項目一覧

No.	イベント	アクション
E1	追加	パッケージの追加ダイアログが起動。DAO・DTO クラス群を出力するパッケージ名を入力し OK ボタンを押下すると、新たなパッケージが追加される。
E2	削除	選択したパッケージが削除される。

表 2-2 イベント一覧

2.2. プロジェクト基本情報の編集と操作

プロジェクト基本情報の編集や、DAO・DTO の設定情報を作成/編集するには、DAO 生成設定ファイルを開く必要があります。DAO 生成設定ファイルは、『新規 DAO 生成設定ファイル作成』ウィザードで指定した「基準ディレクトリー」の直下に、「プロジェクト名.dgd」というファイル名で存在します。DAO 生成設定ファイルを開くと、次のような画面が起動します(レイアウトは使用しているパースペクティブに依存します)。

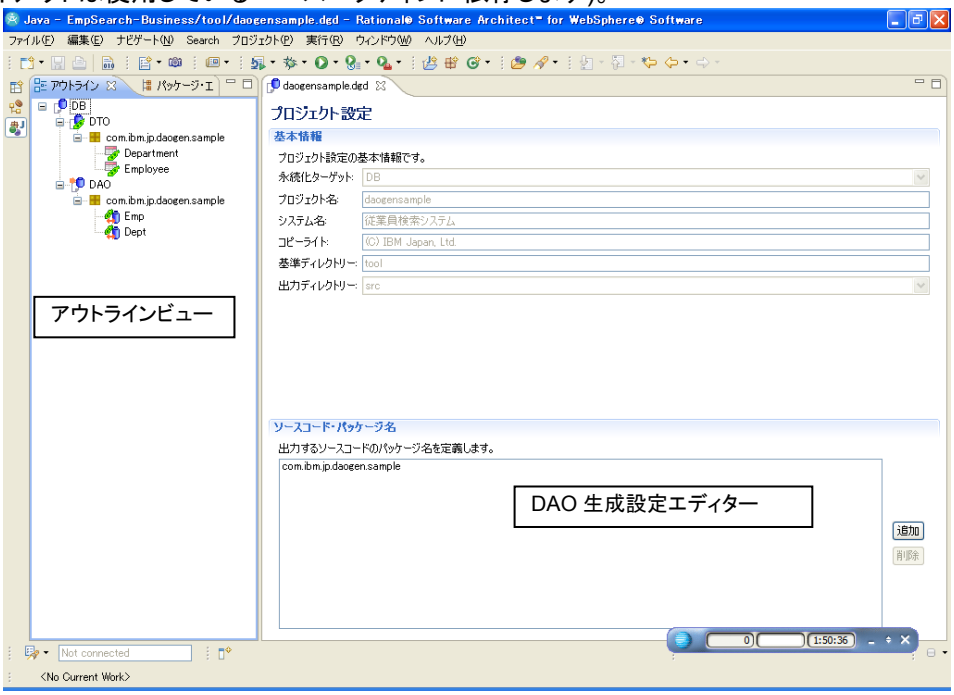


図 2-3 DAO 生成ツールの起動画面

『DAO 生成設定』エディターでは、パッケージ名の追加と削除が可能です。必要に応じて変更してください。その他の設定を変更することはできません。

『アウトライン』ビューには、プロジェクトに登録済みの DTO/DAO 定義情報がパッケージ単位でツリー表示されます。『DAO 生成設定』エディター上では、DTO/DAO 定義情報の追加、および、編集画面の呼び出しを行うことが可能です。

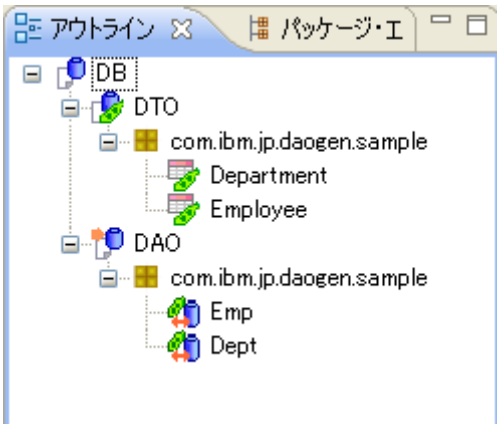


図 2-4 アウトラインビュー

「DTO 定義」には、上部のプルダウンで選択されたパッケージに含まれる、DTO 定義情報の一覧が表示されます。
「DAO 定義」には、同パッケージに含まれる、DAO 定義情報の一覧が表示されます。
各設定情報の追加方法、編集方法については以降の章をご覧ください。

3. DTOの設定

DAO 生成ツールでは、「DTO 定義情報」をもとに DTO クラスの生成を行います。DTO 定義情報は、生成する DTO の数(通常は表の数)だけ作成する必要があります。この章では、DTO 定義情報の作成方法・設定方法について説明します。

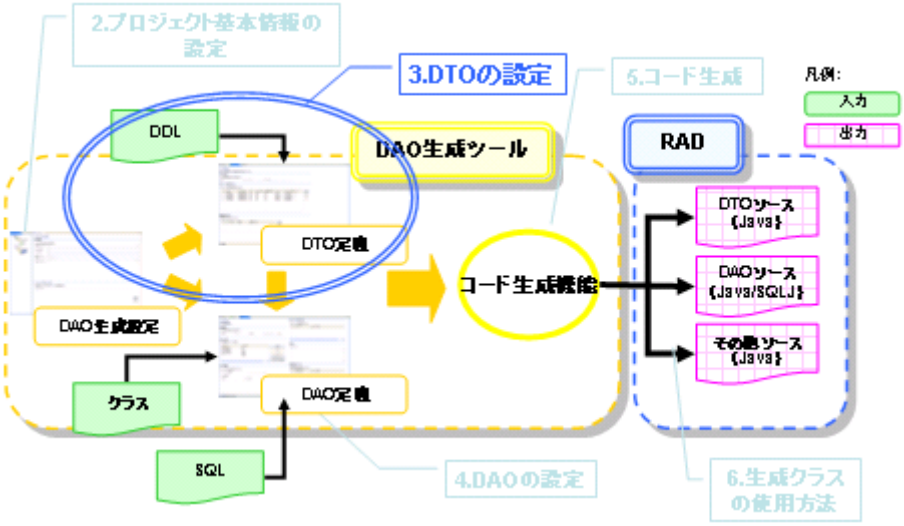


図 3-1 DTO の設定

なお、DTOクラスが既に存在する場合は、DTO定義情報を作成する必要はありません。したがって、この章の内容を省略し、「4 DAOの設定」を直接参照されても構いません。

3.1. DTO定義情報の新規作成

DAO 生成ツールを使用して DTO クラスを生成するには、「DTO 定義」を新規作成し、「DTO 定義情報」を入力する必要があります。DTO 定義情報には DTO のフィールド情報、DTO の各フィールドに対応する列名、およびデータ型、他の DTO に対する参照などの情報が含まれます。通常、「DTO 定義」は DDL 文(CREATE TABLE 文)から作成します。DDL 文はテキスト・エディター、DBMS に付属のツール、RAD 付属のデータ・モデリング・ツールなど、任意の方法により作成してください。「DTO 定義」の新規作成手順は次のとおりです。

1. 『DTO 定義追加』ダイアログの起動
2. DDL 文のインポート

『DTO 定義追加』ダイアログは、アウトラインビューの中から DTO 定義を追加したい DTO パッケージを選択し、右クリック『新規作成』メニューを選択すると起動します。

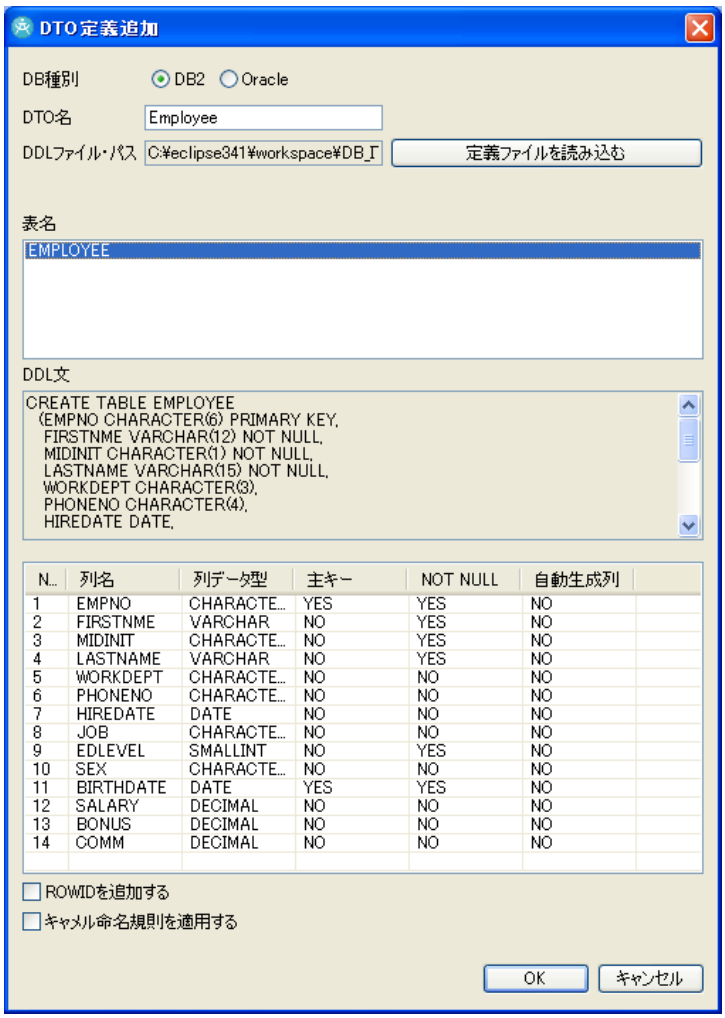


図 3-2『DTO 定義』ダイアログ

『DTO 定義追加』ダイアログから DTO 定義を作成するには、DB 種別を選択し、定義ファイルを読み込む必要があります。DB 種別は、DB2/Oracle から選択してください。定義ファイルを読み込むには、『定義ファイルを読み込む』ボタンを押下し、DDL 文を含むファイルを選択して「OK」ボタンを押下します。インポートした DDL 文の内容がダイアログ上に表示されます。内容を確認の上、「OK」ボタンを押下してください。複数のファイルから同時にインポートすることも可能です。また、1 つのファイルに複数の DDL 文が含まれていてもインポートすることが可能です。インポート可能な DDL 文の書式は次のとおりです。複数の DDL 文を 1 つのファイルに記述する際は、「; (セミコロン)」で区切ります。

```
CREATE TABLE tableName (
columnName1 dataType [PRIMARY KEY],
columnName2 dataType [NOT NULL],
    ・・・・・・
columnNameN dataType [NOT NULL]
)
```

（注意 1）
DTO 名/表名は、スキーマ名に関わらず、それぞれパッケージ内で一意である必要があります。
（注意 2）
DB 種別は 1 つのプロジェクト基本情報につき DB2/Oracle のいずれか一方のみが設定されることを想定しています。

『DTO 定義追加』ダイアログから DTO 定義を作成するには、DB 種別を選択し、定義ファイルを読み込む必要があります。DB 種別は、DB2/Oracle から選択してください。定義ファイルを読み込むには、『定義ファイルを読み込む』ボタンを押下し、DDL 文を含むファイルを選択して「OK」ボタンを押下します。インポートした DDL 文の内容がダイアログ上に表示されます。内容を確認の上、「OK」ボタンを押下してください。複数のファイルから同時にインポートすること

も可能です。

「ROWID を追加する」にチェックをつけた場合、DTO 定義作成時に「ROWID」という名称の列および対応するフィールドが追加されます。これにより、通常の DTO に対して取得した DBMS の ROWID 値をマッピングすることが可能となります。反対に追加しなかった場合、「ROWID」列が DTO 定義に含まれないため、通常の DTO ではなく java.util.Map 型の DTO へマッピングしなければなりません。

通常は、テーブル名の先頭の文字以外を小文字にしたものが DTO クラス名、列名を全て小文字にしたものがそのフィールド名となります。さらに「キャメル命名規則を適用する」にチェックをつけた場合、DTO 定義作成時にそのクラス名とフィールド名が次の規則にしたがって変換されます。

- 「_(アンダースコア)」は削除される
- 「_(アンダースコア)」直後の文字は大文字となる

3.2. 読み込み可能なDDL

3.2.1. CREATE TABLE

DAO 生成ツールが想定している CREATE TABLE 文の書式は次のとおりです。

```
CREATE TABLE [schemaname.][tablename]
        + CONSTRAINT [constraintname] PRIMARY KEY ( column1, column2 ･･･)
        + PRIMARY KEY ( column1, column2 ･･･)
        + [ColumnName] [DataType(size, size)] NOT NULL PRIMARY KEY
        + [ColumnName] [DataType(size, size)] NOT NULL
        + [ColumnName] [DataType(size, size)] PRIMARY KEY
        + [ColumnName] [DataType(size, size)] ･･･ NOT NULL ･･･ PRIMARY KEY ･･･
        + [ColumnName] [DataType(size, size)] NULL
        + [ColumnName] [DataType(size, size)]
        + [ColumnName] [DataType(size)] NOT NULL PRIMARY KEY
        + [ColumnName] [DataType(size)] NOT NULL
        + [ColumnName] [DataType(size)] PRIMARY KEY
        + [ColumnName] [DataType(size)] ･･･ NOT NULL ･･･ PRIMARY KEY
        + [ColumnName] [DataType(size)] NULL
        + [ColumnName] [DataType(size)]
        + [ColumnName] [LONG] [VARGRAPHIC/VARCHAR] NOT NULL PRIMARY KEY
        + [ColumnName] [DataType] NOT NULL PRIMARY KEY
        + [ColumnName] [DataType] GENERATED ALWAYS
        + [ColumnName] [DataType] GENERATED BY DEFAULT
        + [ColumnName] [LONG] [VARGRAPHIC/VARCHAR] NOT NULL
        + [ColumnName] [DataType] NOT NULL
        + [ColumnName] [LONG] [VARGRAPHIC/VARCHAR] PRIMARY KEY
        + [ColumnName] [DataType] PRIMARY KEY
        + [ColumnName] [DataType] ･･･ NOT NULL ･･･ PRIMARY KEY ･･･
        + [ColumnName] [LONG] [VARGRAPHIC/VARCHAR] NULL
        + [ColumnName] [LONG] [VARGRAPHIC/VARCHAR]
        + [ColumnName] [DataType] NULL
        + [ColumnName] [DataType]
```

3.2.2. ALTER TABLE

DAO 生成ツールが想定している ALTER TABLE 文の書式は次のとおりです。なお、同一の表名をもつ DTO 定義が存在しない場合は ALTER TABLE 文を適用することはできません。

```
ALTER TABLE [tablename]
        +ALTER COLUMN [ColumnName] SET DATA TYPE CHAR
        +ALTER COLUMN [ColumnName] SET DATA TYPE CHARACTER
        +ALTER COLUMN [ColumnName] SET DATA TYPE VARCHAR
        +ALTER COLUMN [ColumnName] SET DATA TYPE VARGRAPHIC
        +ADD COLUMN [ColumnName] [DataType (size)]
        +ADD COLUMN [ColumnName] [DataType]
```

```
+ADD CONSTRAINT [ConstraintName] PRIMARY KEY([ColumnName])
+ADD PRIMARY KEY([ColumnName])
+ALTER COLUMN [ColumnName] SET GENERATED ALWAYS
```

3.2.3. CREATE VIEW

DAO 生成ツールが想定している CREATE VIEW 文の書式は次のとおりです。なお、CREATE VIEW によって読み込んだ取得列のデータ型は全て VARCHAR 型となります。

```
CREATE [OR REPLACE] VIEW [schemaname.][viewname]
        + ( colum1, colum2 ･･･) ････ AS SELECT (colum1, column2 ･･･) ･･･
```

3.2.4. 使用可能なJavaデータ型と列データ型

DAO 生成ツール上で使用可能な Java データ型と列データ型は次の通りです。

Java データ型	DB2 データ型	Oracle データ型
Long	BIGINT	BIGINT
byte[]	BLOB	BLOB, RAW
InputStream	BLOB	BLOB
String	CHAR, CHARACTER, VARCHAR, GRAPHIC, VARGRAPHIC, LONG VARCHAR, LONG VARGRAPHIC, CLOB	CHAR, NCHAR, VARCHAR2, NVARCHAR2, CLOB, NCLOB, ROWID
Reader	CLOB	CLOB, NCLOB
java.sql.Date	DATE	
java.util.Date		
BigDecimal	DECIMAL, DEC, NUMERIC, NUM	NUMBER, DECIMAL
Double	DOUBLE	
Integer	INTEGER, INT	INTEGER
Float	REAL	REAL
Short	SMALLINT	SMALLINT
Time	TIME	
Timestamp	TIMESTAMP	DATE, TIMESTAMP

3.3. DTO定義情報の編集と操作

DTO 定義の設定情報を編集するには、『アウトライン』ビューから編集したい DTO 定義のアイコンをダブルクリック、もしくはアイコンを右クリック→「開く」を選択し、『DTO 定義』エディターを起動する必要があります。『アウトライン』ビューは、DTO が属するプロジェクトの『DAO 生成設定』でエディターが起動していないと何も表示されません。『アウトライン』ビューに何も表示されていない場合は、パッケージエクスプローラーなどから DAO 生成設定ファイルをダブルクリックし、『DAO 生成設定』エディターを起動してください。

次に、『DTO 定義』エディターに含まれる、各画面項目とイベントについて説明します。このエディターでは、DTO のフィールド情報や DTO 間の参照関係を編集することが可能です。

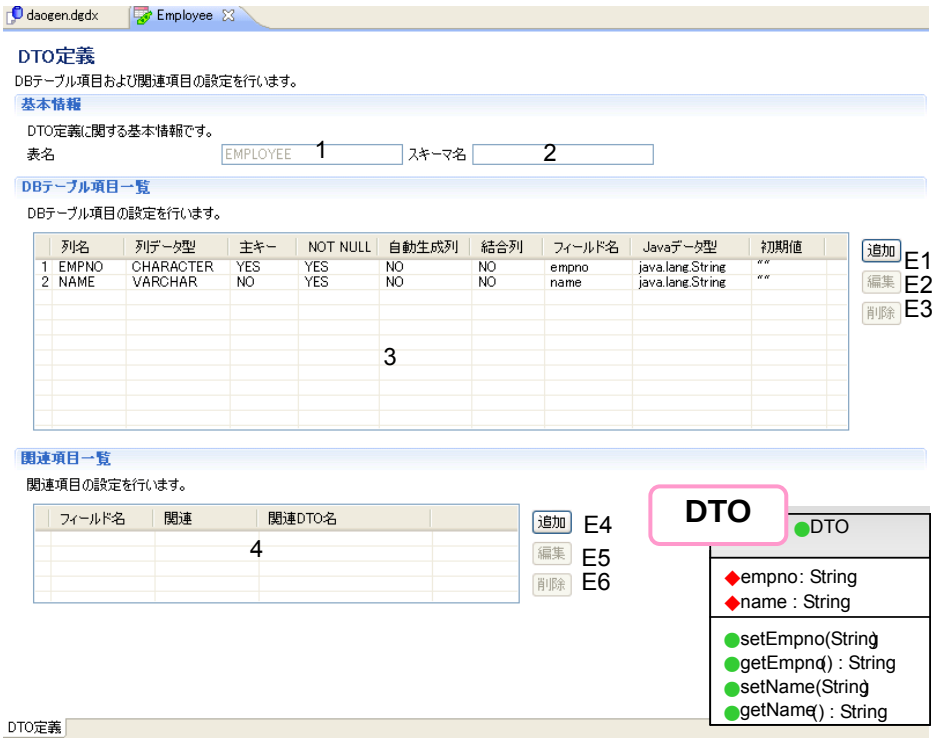


図 3-3『DTO 定義』エディター

No.	項目名	書式	説明
1	表名	半角英数字	現在表示している DTO 定義情報に対応する表名。編集することはできない。ツールによって生成されるメソッド(後述)の SQL 内で使用される。編集不可。
2	スキーマ名	半角英数字	現在表示している DTO 定義情報に対応するスキーマ名。ツールによって生成されるメソッド(後述)の SQL 内で使用される。編集不可。
3	DB テーブル項目一覧	半角英数字	表に含まれる列と DTO が持つフィールドの関係の一覧を表示。各列の意味は後述。
4	関連項目一覧	半角英数字	他の DTO への参照の一覧を表示。JOIN による結果取得時に使用。詳細は後述。

表 3-1 画面項目一覧

No.	イベント	アクション
E1	追加	DB テーブル項目追加ダイアログが起動。各フィールドに値を記入し、「OK」ボタンを押下すると、新たな DB テーブル項目が追加される。
E2	削除	選択した DB テーブル項目が削除される。
E3	編集	DB テーブル項目編集ダイアログが起動。各フィールドに値を記入し、「OK」ボタンを押下すると、DB テーブル項目が更新される。
E4	追加	関連項目追加ダイアログが起動。各フィールドに値を記入し、「OK」ボタンを押下すると、新たな関連項目が追加される。
E5	削除	選択した関連項目が削除される。
E6	編集	関連項目編集ダイアログが起動。各フィールドに値を記入し、「OK」ボタンを押下すると、関連項目が更新される。

表 3-2 イベント一覧

『DTO 定義』エディター上で入力する「DTO 定義情報」と、生成する DTO クラスの間には、1 対 1 の関係があります。『DTO 定義』エディター上で入力する 1 つの「DB テーブル項目」/「関連項目」が、DTO クラスの 1 つのフィールドに対応します。以降では、「DB テーブル項目」と「関連項目」の詳細について説明します。

3.3.1. DBテーブル項目

「DB テーブル項目」には、表の列と DTO のフィールドの関係が表示されます。

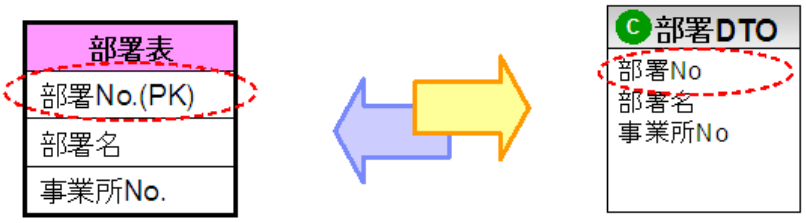


図 3-4 表の列と DTO のフィールドの関係

「DB テーブル項目」に含まれる各属性の意味は次のとおりです。

属性名	意味
No.	自動的に割り振られる連番。
列名	DB の列名。
列データ型	DB 上のデータ型。
主キー	主キーである場合は「YES」。**「YES」である場合は、ツールによっていくつかのメソッド(後述)が出力される。**
NOT NULL	非 NULL である場合は「YES」。
自動生成列	GENERATED ALWAYS 指定がある場合は「YES」。DB2 のみ有効。ツールが生成するメソッド(後述)において、該当列が更新対象となるかどうかを指定。「YES」である場合、その SQL のうち、UPDATE 文と INSERT 文の更新対象列から該当列が除外される。
結合列	結合列である場合は「YES」。V7.1.1 から、JOIN により取得した列を格納するためのフィールドとして結合列が追加されました。JOIN により取得した列を格納するために DTO の包含関係で取得する方法や、Map として取得する方法、DTO を別途作成する方法がありますが、それ以外に、結合列を追加することによる対応が可能となりました。結合列として定義したフィールドは、デフォルト CRUD メソッドの取得列には含まれず、更新用メソッドのマッピング・エディターには表示されません。
フィールド名	DTO が持つフィールド名。列とフィールドは 1 対 1 に対応する。
Java データ型	DTO が持つフィールドの Java クラス名。列データ型を指定すると、自動的に入力される。
初期値	DTO が持つフィールドの初期値。値を明示的に指定しない場合は、初期値に指定した値が各フィールドのデフォルト値となる。

表 3-3 「DB テーブル項目」属性一覧

DB テーブル項目の追加/編集/削除は、「DB テーブル項目一覧」右部のボタンを使用して行ないます。『追加(編集)』ボタンを押下すると、『DB テーブル項目追加(編集)』ダイアログが起動します。ダイアログに各属性の値を入力し、「OK」ボタンを押下すると、DB テーブル項目が追加(編集)されます。

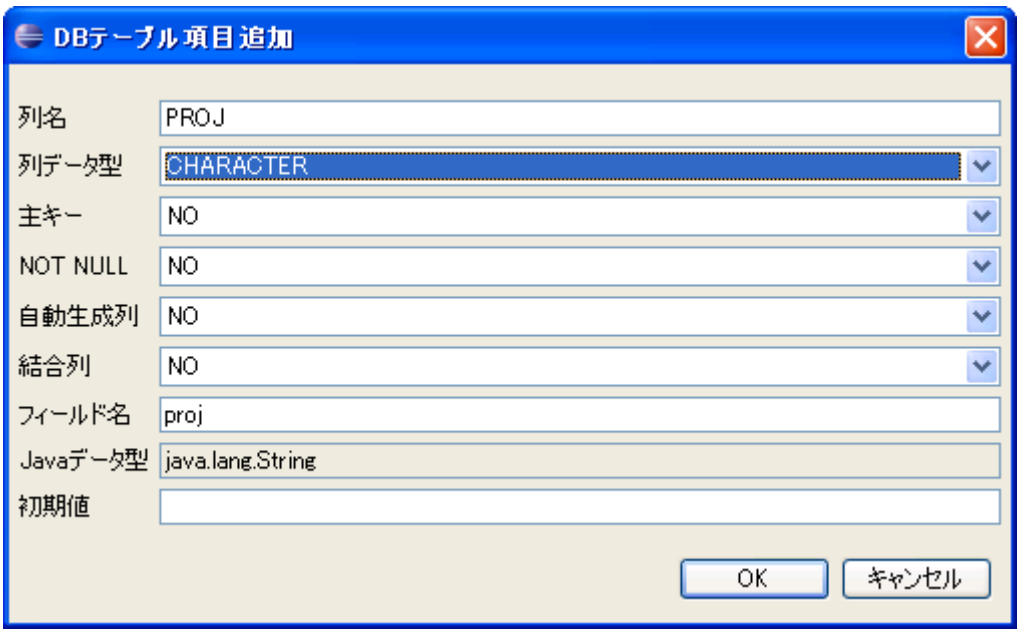


図 3-5『DB テーブル項目追加』ダイアログ

なお、DAO 生成ツールでは、**DTO クラスに primitive 型(int, double など)のフィールドを追加することはできません。**
これは、Java の primitive 型には null が無く、DB 上の NULL を表現することができないためです。

3.3.2. 関連項目

「関連項目」には、表の関連と DTO 参照の関係が表示されます。後述する DAO が持つメソッドによって、**JOIN 結果を DTO の包含関係として取得する場合は、事前に JOIN する表の関連を「関連項目」として入力しておく必要があります。**

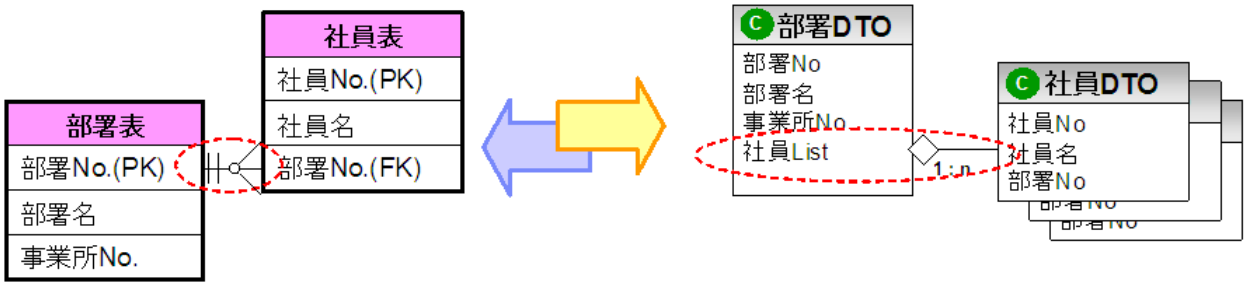


図 3-6 表の関連と DTO 参照の関係(図は 1:n 関連)

「関連項目」に含まれる各属性の意味は次のとおりです。なお、JOIN結果の取得方法の詳細については「4 DAOの設定」をご覧ください。

属性名	意味
No.	自動的に割り振られる連番。
フィールド名	DTO が持つフィールド名。
関連	「関連表名」に指定した表との関係。「1:1」または「1:n」から選択。「1:1」を選択した場合は、指定した「関連表名」に対応する DTO の 1 インスタンスを、1 つのフィールドとして保持。「1:n」を選択した場合は、同 DTO の複数のインスタンスを List 型のフィールドとして保持。
関連表名	関連先の表名。

表 3-4 「関連項目」属性一覧

関連項目の追加/編集/削除は、「関連項目一覧」右部のボタンを使用して行ないます。『追加(編集)』ボタンを押下すると、『関連項目追加(編集)』ダイアログが起動します。ダイアログに各属性の値を入力し、「OK」ボタンを押下すると、関連項目が追加(編集)されます。

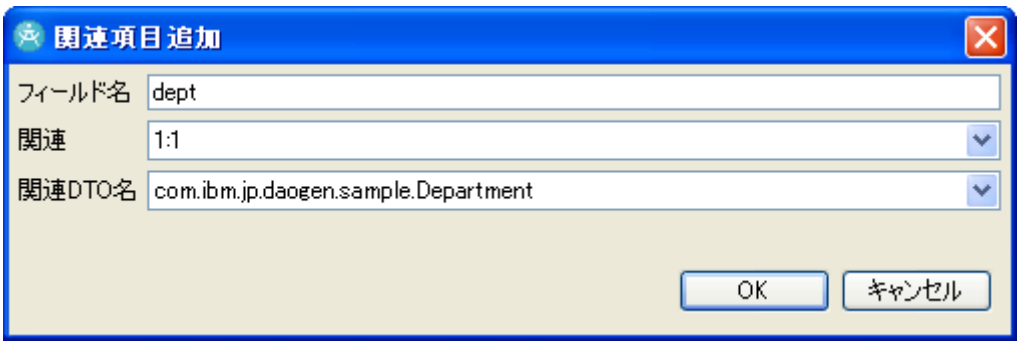


図 3-7『関連項目追加』ダイアログ

4. DAOの設定

DAO 生成ツールでは、「DAO 定義情報」をもとに DAO クラスの生成を行います。DAO 定義情報には DAO のメソッド情報、使用する SQL、SQL の取得列/パラメーターと DTO のフィールドの対応関係などの情報が含まれます。また、DAO 定義情報は、生成する DAO の数(通常は表の数)だけ作成する必要があります。この章では、DAO 定義情報の作成方法・設定方法について説明します。

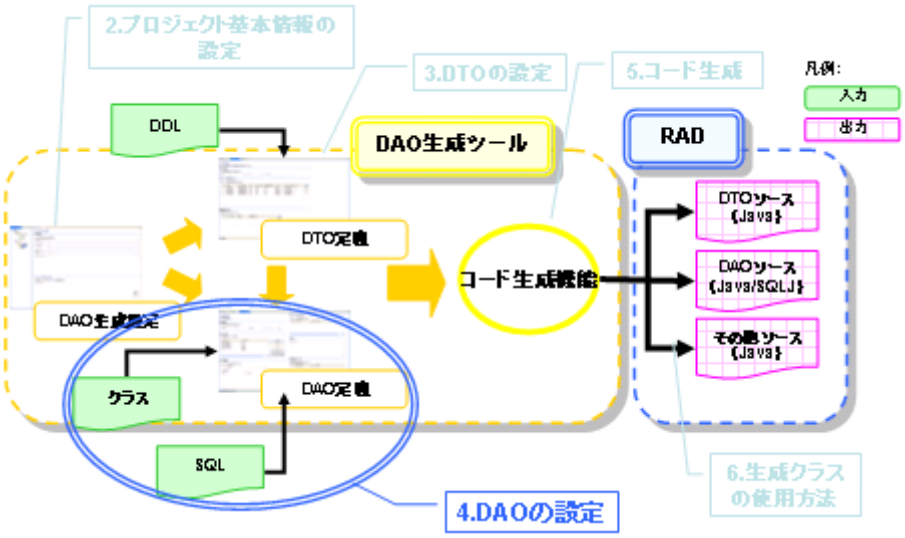


図 4-1 DAO の設定

なお、DAO 定義情報を DTO 定義情報から作成する場合と DTO クラスから作成する場合とでは、新規作成時の操作が異なります。また、**DTO 定義情報から作成した DAO 定義情報と、DTO クラスから作成した DAO 定義情報を混在させることはできません。**

4.1. DAO定義情報の新規作成

DTO 定義情報から DAO 定義情報を作成する場合は、アウトラインビューの中から定義情報を追加したい DAO パッケージを選択し、右クリックのメニューから『DTO 定義から DAO 定義を追加』を選択し、『DAO 定義追加』ダイアログを起動してください。

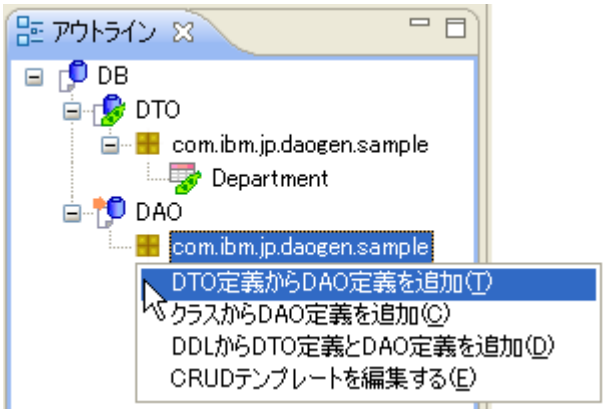


図 4-2 『DTO 定義から DAO 定義を追加』メニュー

『DAO 定義追加』ダイアログでは、「DAO 名」を入力し、対応する表名をプルダウンから選択します。**「DAO 名」は生成する DAO クラス、および、その他関連するクラスの名前として使用されます。**「OK」ボタンを押下すると、「DAO 定義情報」が新規作成されます。

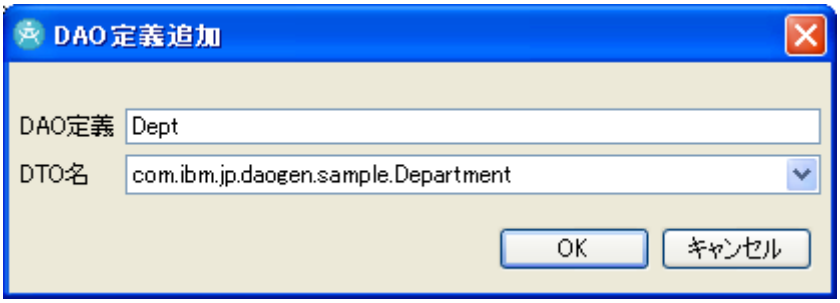


図 4-3『DAO 定義追加』ダイアログ

DTO クラスから DAO 定義情報を作成する場合は、アウトラインビューの中から定義情報を追加したい DAO パッケージを選択し、右クリックのメニューから『クラスから DAO 定義を追加』を選択し、『DAO 定義追加』ダイアログを起動してください。

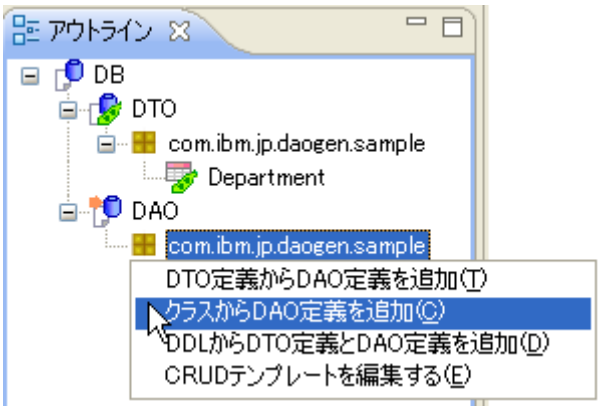


図 4-4『クラスから DAO 定義を追加』メニュー

『DAO 定義追加』ダイアログでは、次の情報を入力し、「OK」ボタンを押下して下さい。

1. 対象 RDBMS：「DB2」または「Oracle」
2. DTO クラス名：『参照』ボタンを押下し、任意の DTOJava クラスを選択します。
3. DAO 定義：任意の名前を入力します。

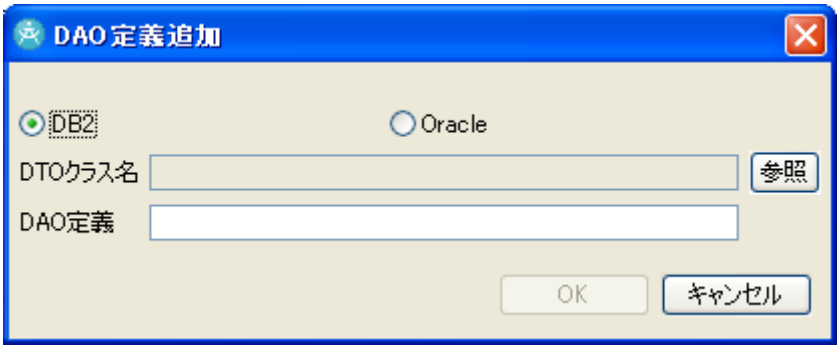


図 4-5『DAO 定義追加』ダイアログ

4.2. DAO定義情報の編集と操作

DAO の設定情報を編集するには、『アウトライン』ビューから編集したい DAO 定義のアイコンをダブルクリックし、『DAO 定義』エディターを起動する必要があります。『アウトライン』ビューに何も表示されていない場合は、パッケージエクスプローラーなどから DAO 生成設定ファイルをダブルクリックし、『DAO 生成設定』エディターを起動してください。次に、『DAO 定義』エディターに含まれる、各画面項目とイベントについて説明します。このエディターでは、DAO のメソッド情報や使用する SQL、および、SQL の取得列/パラメーターと DTO フィールドの対応関係を編集することが可能です。当資料では、照会を行なう SQL を『照会系』、挿入/更新/削除を行なう SQL を『更新系』として表現しています。『照会系』、『更新系』それぞれ、表示される項目や動作が異なります。

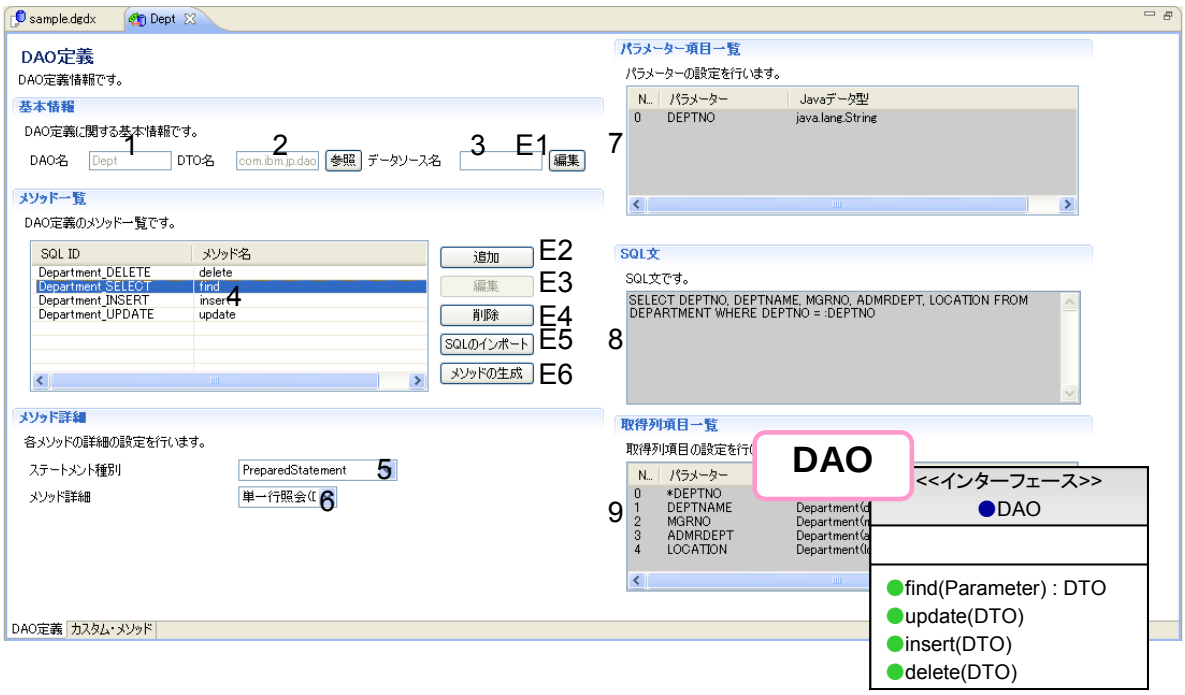


図 4-6 『DAO 定義』エディター

No.	項目名	書式	説明
1	DAO 名	半角英数字	新規作成時に入力した DAO 名を表示。編集不可。
2	DTO 名	半角英数字	新規作成時に選択した DTO 名を表示。
3	データソース名	半角英数字	複数のデータソースを使用する場合、表示中の DAO と関連するデータソース名を指定。ただし、この指定は Spring 実装のみ有効。
4	メソッド一覧	半角英数字	SQL-ID(SQL 名)、および、SQL-ID に対応する、DAO が持つメソッドの一覧。メソッド名は、SQL-ID に基づき、自動的に名前が付加される。メソッド名は、セルをダブルクリックすることにより直接変更することが可能。**ただし、単一行挿入・照会・更新・削除メソッド(後述)は編集することができない。**
5	ステートメント種別	プルダウン(Statement/ PreparedStatement)から選択	DAO 内部での SQL 発行時に、java.sql.Statement インターフェースを使用するか、java.sql.PreparedStatement インターフェースを使用するかを選択。デフォルトは「PreparedStatement」。**ただし、通常は変更する必要は無い。**また、DAO の実装として SQLJ/Spring(SQLJ)を選択した場合、ステートメント種別の選択は無効である。
6	メソッド詳細	プルダウン	選択した SQL(SQL-ID)に対応する、DAO が持つメソッドの引数や戻り値を変更する。詳細は後述。
7	パラメーター項目	プルダウン/ダイアログ起動	選択した SQL(SQL-ID)内に含まれるパラメーター(ホスト変数)の一覧を表示。『照会系』の場合、パラメーターのクラス名をプルダウンから選択する。『更新系』、かつ、メソッド詳細上で「Parameter 指定更新(INSERT SELECT」が選択されている場合は、『照会系』と同様。そうでない場合は、パラメーターと DTO フィールドの対応関係を定義するダイアログ(DTO へのマッピングダイアログ)が起動する。詳細は後述。
8	SQL 文	半角英数字	選択した SQL-ID に対応する SQL 文を表示。

No.	項目名	書式	説明
9	取得列項目	プルダウン/ダイアログ起動	選択した SQL(SQL-ID)内に含まれる取得列の一覧を表示。『更新系』の場合は表示されない。メソッド詳細上で「Map DTO」が選択されている場合は、取得結果列のクラス名をプルダウンから選択する。そうでない場合は、取得列と DTO フィールドの対応関係を定義するダイアログが起動する。詳細は後述。

表 4-1 画面項目一覧

No.	イベント	アクション
E1	参照	『DTO クラス名』ダイアログが起動。DAO 定義に対応する DTO クラス名を変更する。クラスから DAO 定義を作成した場合のみ有効。
E2	追加	『SQL 定義の追加』ダイアログが起動。SQL-ID、SQL 文を入力し、『OK ボタンを押下すると、SQL が追加される。
E3	編集	『SQL 定義の編集』ダイアログが起動。SQL 文を入力し、『OK ボタンを押下すると、その取得列およびパラメーターが、「パラメーター項目」および「取得列項目」に反映される。
E4	削除	選択した SQLID が削除される
E5	SQL のインポート	『SQL のインポート』ダイアログが起動。SQL が含まれる外部ファイルを選択し、「OK」ボタンを押下すると、SQL がインポートされ、対応するメソッドが追加される。
E6	メソッドの生成	CRUD テンプレートを元にメソッドを再生成する。DTO 定義を編集した場合等に使用する。

表 4-2 イベント一覧

『DAO定義』新規作成時で、かつ、**対応する表に主キー列が存在する場合**、主キー条件を含むINSERT文、SELECT文、UPDATE文、DELETE文が自動的に生成され、それぞれDAOが持つメソッドとして『DAO定義』に追加されます。デフォルトでは、次のメソッド(SQL-ID)がDAO定義に追加されます。各メソッドの詳細は「6 生成クラスの使用」をご覧ください。
※*DAOName* は、DAO 定義を追加するときに指定した「DAO 名」を表しています。

メソッド名	SQL-ID	説明
find	*DAOName*_SELECT	主キー条件を持ち、表の全列を取得する SELECT 文を発行。
insert	*DAOName* _INSERT	自動生成列を除く、表の全列に値を挿入する INSERT 文を発行。
update	*DAOName* _UPDATE	主キー条件を持ち、主キーおよび自動生成列以外の全列を更新する UPDATE 文を発行。
delete	*DAOName* _DELETE	主キー条件を持つ DELETE 文を発行。

表 4-3 生成されるメソッド

『DAO 定義』新規作成時、または、「メソッドの生成」ボタン押下時に生成されるメソッドの種類は、「CRUD テンプレートを編集する」メニューから、パッケージ毎にカスタマイズすることが可能です。

図 4-7『CRUD テンプレートを編集する』メニュー

例えば、照会用 SQL の WHERE 句の後部に削除フラグの有無を判定する条件を追加することで、論理削除されたレコードを照会しない SQL およびメソッドを生成することができます。テンプレートの書式は次の通りです。

```
[基本書式]
[SQL-ID].[Statement Type].[Method Type].[MethodName] = [SQL Template]

[Statement Type]
ST      :java.sql.Statement
PS      :java.sql.PreparedStatement

[Method Type]
DI      :(デフォルト)単一行挿入
DS      :(デフォルト)単一行照会(0 件時例外)
DF      :(デフォルト)単一行照会
DU      :(デフォルト)単一行更新
DD      :(デフォルト)単一行削除
SS      :単一行照会(0 件時例外)
SF      :単一行照会
MF      :複数行照会
LF      :複数行照会(範囲指定あり)
SSM     :単一行照会(Map/0 件時例外)
SFM     :単一行照会(Map)
MFM     :複数行照会(Map)
LFM     :複数行照会(Map/範囲指定あり)
SI      :単一行挿入
PI      :複数行挿入
BI      :バッチ挿入
SU      :単一行更新
PU      :複数行更新
BU      :バッチ更新
SD      :単一行削除
PD      :複数行削除
BD      :バッチ削除

[SQL Template]
$TABLE                      :テーブル名                  例  Employee
$PRIMARYKEY_CONDITION       :PK を使用した WHERE 条件    例  EMPNO = :EMPNO
$VARIABLE_CONDITION         :全ての条件の列挙            例  EMPNO LIKE :EMPNO
$INSERT_COLUMNS             :GK を除いたカラム名の列挙   例  EMPNO, FIRSTNME....
$INSERT_VALUES              :GK を除いたパラメーターの列挙 例 :EMPNO, :FIRSTNME ....
$SELECT_COLUMNS             :すべてのカラム名の列挙      例  EMPNO, FIRSTNME ....
$UPDATE_VALUES              :PK、GK を除いた条件の列挙   例  FIRSTNME = :FIRSTNME

[設定例]
INSERT.PS.DI.insert = INSERT INTO $TABLE ($INSERT_COLUMNS) VALUES($INSERT_VALUES)
SELECT.PS.DF.find = SELECT $SELECT_COLUMNS FROM $TABLE WHERE $PRIMARYKEY_CONDITION
UPDATE.PS.DU.update = UPDATE $TABLE SET $UPDATE_VALUES WHERE $PRIMARYKEY_CONDITION
DELETE.PS.DD.delete = DELETE FROM $TABLE WHERE $PRIMARYKEY_CONDITION
```

『DAO 定義』エディター上で入力する「DAO 定義情報」と、生成する DAO クラスの間には、1 対 1 の関係があります。『DAO 定義』エディター上で入力する 1 つの「SQL-ID(メソッド名)」が、DAO クラスの 1 つのメソッドに対応します。なお、「JDBC 実装」の場合 SQL はプロパティー・ファイル内に集約されるため、「SQL-ID」はパッケージにつき一意である必要があります。以降では、メソッドの追加方法と編集方法について説明します。

4.2.1. ユーザー定義CRUDメソッドの追加

前述したツールによって生成されるメソッド以外に、『DAO 定義』作成後に追加するメソッドを、DAO 生成ツールでは「ユーザー定義 CRUD メソッド」と呼びます。「ユーザー定義 CRUD メソッド」を追加するには、次の 3 通りの方法があります。

方法 1:エディター上から追加
方法 2:外部ファイルから追加
方法 3:カスタム・メソッドとして追加

以降では、各方法の詳細を説明します。

エディター上から追加

『DAO定義』エディター上の「追加」ボタンを押下し、『SQLの追加』ダイアログを起動します。ダイアログ上で「SQL-ID」と「SQL文」を入力し、「OK」ボタンを押下すると、エディター上にSQL-IDおよびメソッドが追加されます。使用可能なSQLの書式については「4.2.2 使用可能なSQL」をご覧ください。サポートしていない書式(ストアド・プロシージャーなど)を使用する場合は、「SQLのみ追加」にチェックを入れ、「OK」ボタンを押下してください。

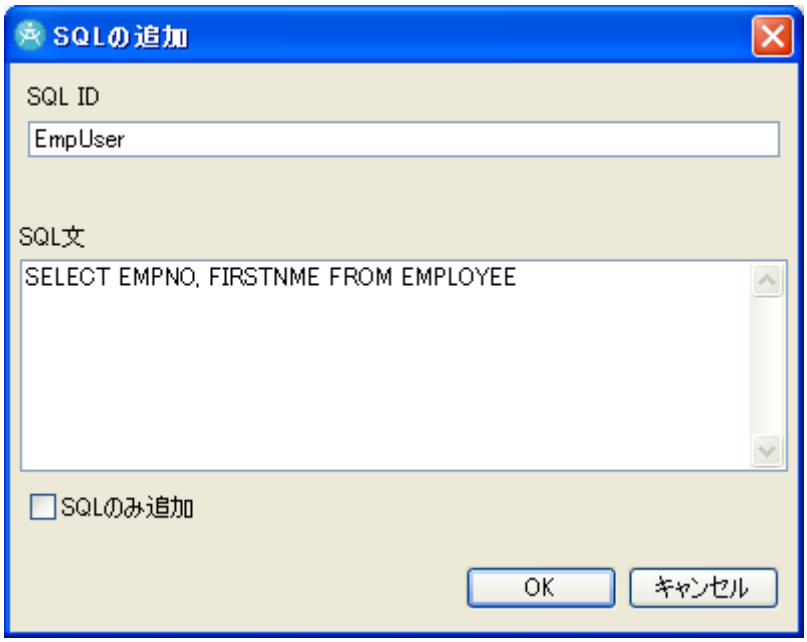


図 4-8 SQL 定義の追加

外部ファイルから追加

SQL を外部からインポートすることによってメソッドを追加することも可能です。まず、次の形式のテキスト・ファイル(SQL 定義ファイル)を用意してください。書式は java.util.Properties クラスの仕様を拡張したものです。インポート時に使用する文字コード(エンコード)は、Java プロジェクトに指定されているものを使用します(MS932 など)。

ファイル名		
*.properties		
書式		
SQL-ID[.*stmtType*] = *SQL*		
キーワード	**値**	**内容**
SQL-ID	任意の文字列	SQL を識別するためのキー。
stmtType	「ST」/「PS」	「ST」を選択した場合は、java.sql.Statement インターフェースにより SQL が発行され、「PS」の場合は java.sql.PreparedStatement インターフェースが使用されます。省略時は「PS」となります。
SQL	任意文字列	SQL文。使用可能なSQLの書式については「4.2.2 使用可能なSQL」を参照。改行など特殊文字の扱いはjava.util.Propertiesの仕様に準ずる。

表 4-4 SQL 定義ファイルの書式

```
(SQL 定義ファイルの記述例)
#コメント
Emp_SelectByName.PS = SELECT EMPNO, NAME FROM EMP WHERE NAME LIKE :name
Emp_UpdatePhoneno = UPDATE EMP SET PHONENO = :phoneno WHERE EMPNO = :empno
```

『SQL のインポート』ボタンを押下し、『SQL のインポート』ダイアログを起動します。その後、『定義ファイルを読み込む』ボタンを押下し、続いてファイル・ダイアログから用意した「SQL 定義ファイル」を選択してください。すると、インポート対象 SQL フィールドに、ファイルに含まれる SQL-ID の一覧が表示されます。インポートする SQL-ID にチェックがついていることを確認し、「OK」ボタンを押下すると、ユーザー定義 CRUD メソッドとして SQL-ID がエディター上に追加されます。なお、サポートされていない書式(ストアド・プロシージャーなど)を含む SQL は「その他」に分類され、メソッドは登録されず SQL のみ追加されます。

(注意)
SQL-ID はパッケージ内で一意になるよう設定してください。SQL-ID が重複する場合は、次のような挙動となります。

- SQL 定義ファイルに同じ SQL-ID の SQL を記述した場合は、後に記述してある SQL を採用します。
- 既に DAO 定義に登録されている SQL-ID を読み込んだ場合は、上書きされず既に登録してある SQL-ID が優先されます。

なお、ユーザー定義CRUDメソッドの名前は、SQLの種類、SQL-IDおよび「メソッド詳細設定」に基づいて自動的に作成されます。任意の名称に変更することも可能です。命名規則の詳細は「4.2.3 ユーザー定義CRUDメソッドの編集」をご覧ください。

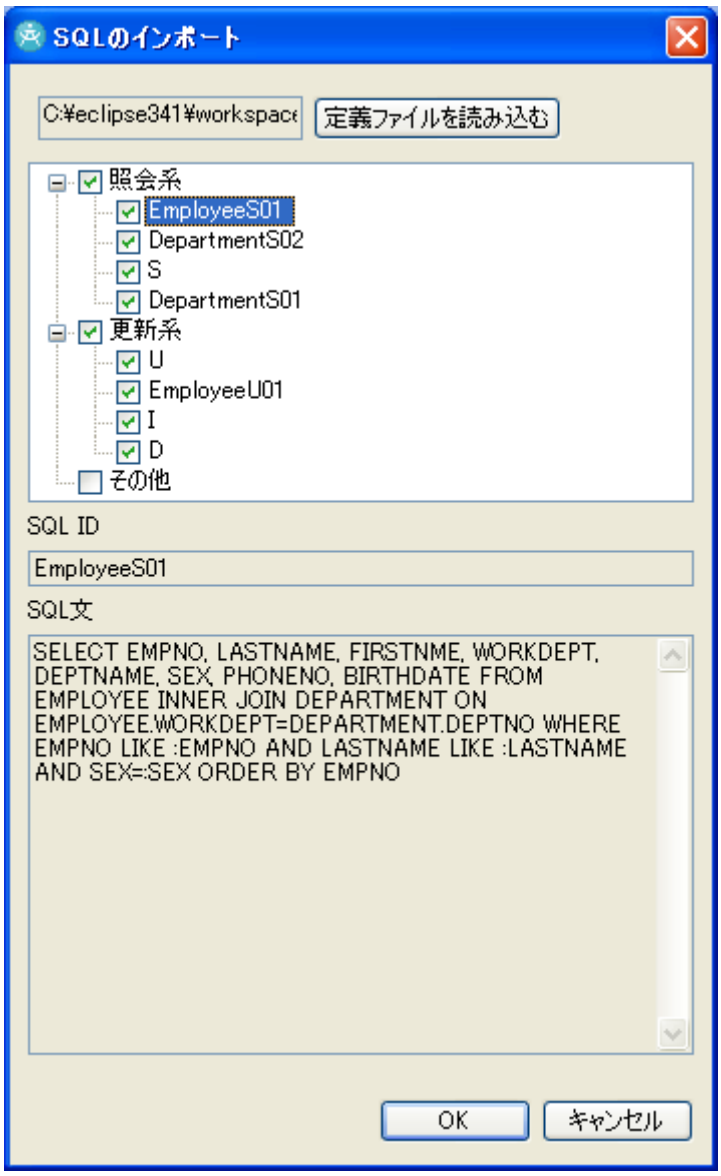


図 4-9『SQL のインポート』ダイアログ

カスタム・メソッドとして追加・編集

DAO にメソッドのインターフェースだけを追加し、メソッドを独自に実装することも可能です。DAO 生成ツールでは、このメソッドを「カスタム・メソッド」と呼びます。例えば、DAO 生成ツールでサポートされていないストアード・プロシージャーの実行で実装する必要があります。

カスタム・メソッドは、『カスタム』エディター上で追加することが可能です。『カスタム』エディターを表示するには、『DAO定義』エディター下部のタブをクリックしてください。なお、カスタム・メソッドの実装方法については「6 生成クラスの使用方法」をご覧ください。

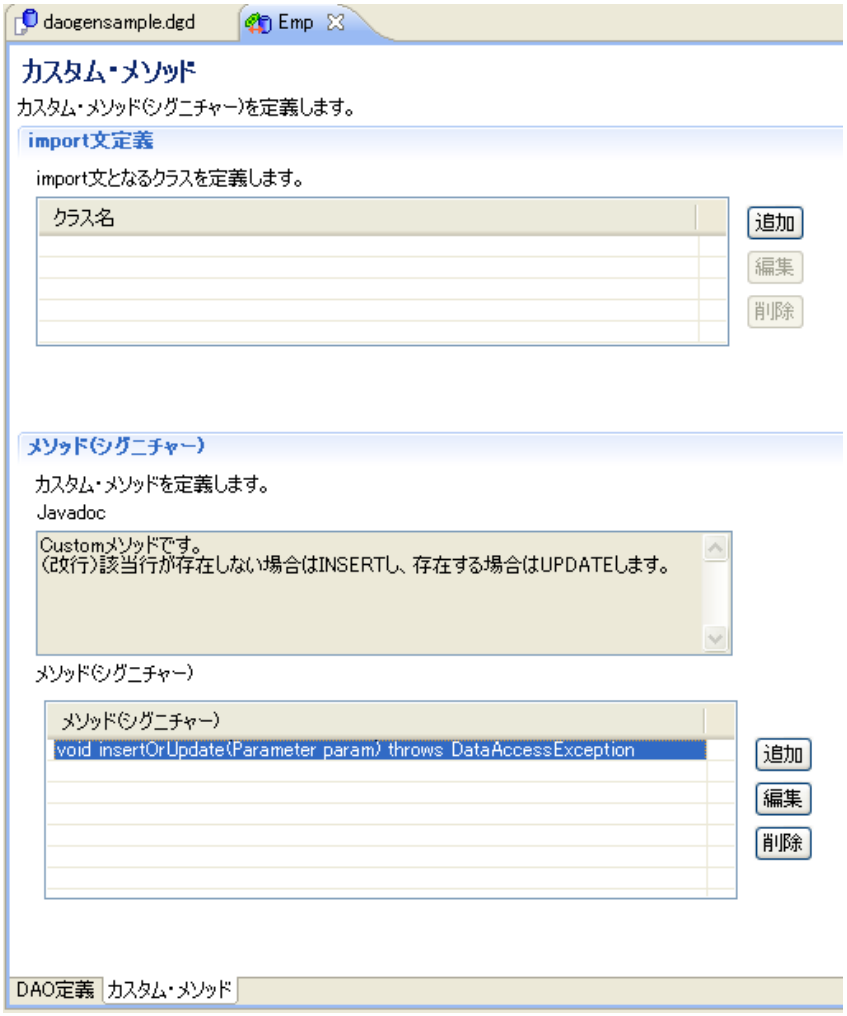


図 4-10『カスタム』エディター

エディター上「追加メソッド」右部の「追加」/「削除」/「編集」ボタンにより、カスタム・メソッドの追加/編集/削除を行うことができます。**追加したカスタム・メソッド(シグニチャーおよび Javadoc)、はそのまま生成コードに反映されるため、Java 言語仕様に沿って入力する必要があります。**

(例)
public void cusomMethod(String arg) throws IllegalArgumentException // 文末の「;(セミコロン)」は不要

また、シグニチャー上で必要なクラスは、エディター上「追加 import 文」右部の「追加」/「削除」/「編集」ボタンによってあらかじめ追加しておく必要があります。

シグニチャー、Javadoc、または、追加 import 文の入力を誤ると、生成コードのコンパイル時にエラーとなる可能性があります。コンパイル・エラーとなった場合はカスタム・メソッドを編集し、正しい書式に修正してください。

4.2.2. 使用可能なSQL

DAO 生成ツールでは、次のルールに従った SQL を使用してください。

パラメーターの記述方法

パラメーターはホスト変数形式(「:(コロン)」+「変数名」)で記述してください。パラメーターは、ステートメント種別として「Statement」を選択した場合は文字列に、「PreparedStatement」を選択した場合は「?」に変換されます。なお、SQLに指定した変数名がパラメーター設定時のキーとなります。コード上でのパラメーター設定方法については「6 生成クラスの使用方法」をご覧ください。

(例)
SELECT EMPNAME FROM EMPLOYEE WHERE EMPNO = :**EMPNO**

関数

SQL で COUNT や AVG 等の関数を使用する場合、AS 句を使用し別名を指定する必要があります。DAO 生成ツールでは別名を取得列名として扱います。**指定しない場合、DAO 生成ツールが誤った取得列名を認識してしまう可能性があります。**

(例)
○ SELECT COUNT(*) **AS** COUNT FROM EMPLOYEE
× SELECT COUNT(*) FROM EMPLOYEE
× SELECT COUNT(*) COUNT FROM EMPLOYEE

動的パラメーター照会機能

・基本機能

ステートメント種別が「ST」または「PS」である場合は、WHERE 文以降の条件を下の例のように動的に変更する事ができます。これを「動的パラメーター照会機能」と呼びます。この機能を利用する場合は「:_WHERE_AND」、「:_WHERE_OR」、「:_WHERE_IN」、「:_OR」、「:_AND」、「:_IN」変数を使用して後続の括弧内に動的に変更したい条件を「,」(カンマ)で列記します。**なお、『新規 DAO 生成設定ファイル作成』ウィザードで、実装種別として「SQLJ」、「Spring(SQLJ)」を選択した場合、この機能を使用することはできません。**

(例)
SQL テンプレート
SELECT EMPNO FROM EMPLOYEE **:_WHERE_AND**(EMPNO = :EMPNO, JOB <> :JOB, SEX = :SEX)
パラメーター名
EMPNO, JOB
パラメーター値
000010, MANAGER

変換後の SQL(ステートメント種別「ST」)
SELECT EMPNO FROM EMPLOYEE WHERE EMPNO = '000010' **AND** JOB <> 'MANAGER'

上の例では3つの条件「EMPNO = :EMPNO」、「JOB <> :JOB」、「SEX = :SEX」が定義されています。この定義に対して指定されたパラメーターが「EMPNO」、「JOB」である場合、条件「EMPNO = :EMPNO」と「JOB <> :JOB」が有効となり、「AND」によって条件が連結されます。「:_WHERE_AND」変数ではなく「:_WHERE_OR」変数を使用した場合は、「OR」によって条件が連結されます。1つの条件に含める事ができる変数は1つだけです。条件の中に変数が存在しない場合は、パラメーターに関わらず常に有効となります。また、「:_WHERE_IN」変数を使用した場合、「IN」句に対して複数の値(条件)を設定することができます。

(例)
SQL テンプレート
SELECT EMPNO FROM EMPLOYEE **:_WHERE_OR**(EMPNO = :EMPNO, JOB <> :JOB)
パラメーター名
EMPNO, JOB
パラメーター値
000010, MANAGER

変換後の SQL(ステートメント種別「ST」)
SELECT EMPNO FROM EMPLOYEE WHERE EMPNO = '000010' **OR** JOB <> 'MANAGER'

また、「:_AND」、「:_OR」および「:_IN」変数でも同様の機能を実現することができます。「:_WHERE_AND」、「:_WHERE_OR」および「:_WHERE_IN」と異なるのは次の 2 点です。

- 条件の有無に関わらず「WHERE」句は出力しない
- 「:_WHERE_AND」、「:_WHERE_OR」、「:_WHERE_IN」、「:_AND」および「:_OR」変数内部で使用することができる(入れ子)

(例)
SQL テンプレート
SELECT EMPNO FROM EMPLOYEE**:_WHERE_OR**(EMPNO = :EMPNO, **:_AND**(JOB <> :JOB, NAME LIKE :NAME))
パラメーター名
EMPNO, JOB, NAME
パラメーター値
000010, MANAGER, MICHAEL

変換後の SQL(ステートメント種別「ST」)
　SELECT EMPNO FROM EMPLOYEE WHERE EMPNO = ‘000010’ **OR** (JOB <> ‘MANAGER’ **AND** NAME LIKE ‘MICHAEL’)

「:_WHERE_IN」変数の場合は、IN 句に含まれる条件が、設定した複数のパラメーター値の数だけ動的に追加されます。また、パラメーターの設定には Parameter # setObject(String, Object, int)を使用する必要があります。

(例)
SQL テンプレート
　SELECT EMPNO FROM EMPLOYEE **:_WHERE_IN**(EMPNO = :EMPNO, ‘1’, ‘2’, ‘3’)
パラメーター名
　EMPNO
パラメーター値
　000010, 000020

```
Parameter param = new Parameter();
param.setObject("EMPNO", new String[]{“000010”, “000020”}, Types.VARCHAR);      // 配列値を設定
DTO dto = dao.findID1(param);
```

変換後の SQL(ステートメント種別「ST」)
　SELECT EMPNO FROM EMPLOYEE WHERE EMPNO IN ('000010', ‘000020’, ‘1’, ‘2’, ‘3’)

・IS NULL 自動変換機能
動的パラメーター照会機能の条件に含まれる比較演算子が「=」であり、かつ、パラメーター値に「null」が設定されている場合は、「=」と変数が文字列「IS NULL」によって置換されます。なお、「:_WHERE_IN」変数の場合は同機能は有効とならず、パラメーターに対して「null」が設定されます。

(例)
SQL テンプレート
　SELECT EMPNO FROM EMPLOYEE :_WHERE_AND(EMPNO = :EMPNO, JOB <> :JOB, SEX = :SEX)
パラメーター名
　EMPNO
パラメーター値
　null

変換後の SQL(ステートメント種別「ST」)
　SELECT EMPNO FROM EMPLOYEE WHERE EMPNO **IS NULL**

・IS NOT NULL 自動変換機能
動的パラメーター照会機能の条件に含まれる比較演算子が「<>」、「!=」または「^=」であり、かつ、パラメーター値に「null」が設定されている場合は、演算子と変数が文字列「IS NOT NULL」によって置換されます。

(例)
SQL テンプレート
　SELECT EMPNO FROM EMPLOYEE :_WHERE_AND(EMPNO = :EMPNO, JOB <> :JOB, SEX = :SEX)
パラメーター名
　JOB
パラメーター値
　null

変換後の SQL(ステートメント種別「ST」)
　SELECT EMPNO FROM EMPLOYEE WHERE JOB **IS NOT NULL**

DTO フィールドと取得列名/パラメーター名の関係
SQL に含まれる取得列名(別名含む)とパラメーター名は、DTO 定義上の同じ列名(**大文字・小文字は区別しません**)を持つフィールドにマッピングされます。

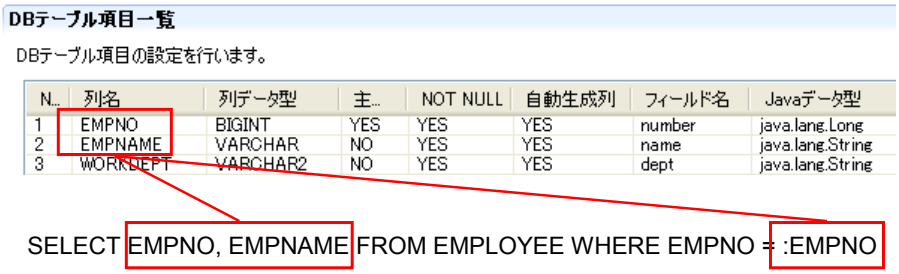


図 4-11 取得列名によるマッピング

また、同じ列名を持つフィールドが無かった場合、DTO 定義上の同じフィールド名(**大文字・小文字を区別します**)を持つフィールドにマッピングされます。

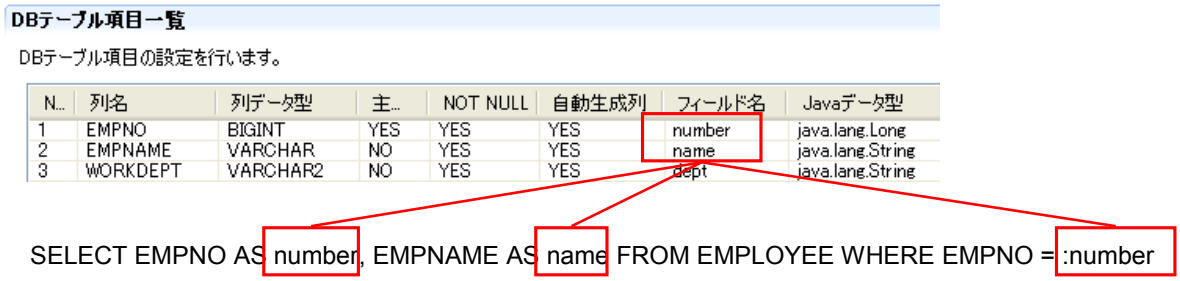


図 4-12 フィールド名による関連

さらに、取得列名またはパラメーターに「_(アンダーバー)」が含まれる場合は、参照先の DTO 定義上の同じフィールド名(**大文字・小文字を区別します**)を持つフィールドにマッピングされます。次の例では、参照先の DTO(定義)「sample.Emp」が持つ「number」フィールドにマッピングされます。

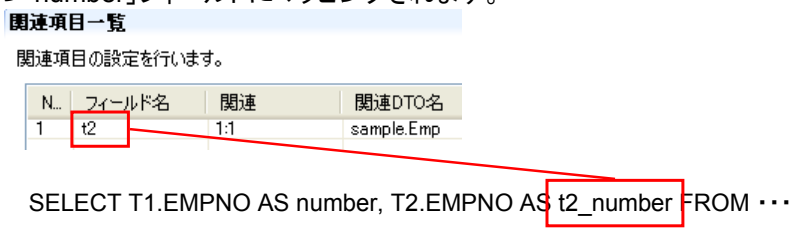


図 4-13 参照先の DTO 定義へのマッピング

以上のルールによってマッピングされない取得列名/パラメーター名は、手動によりマッピングする必要があります。

その他

・SQL 文には改行文字、タブ文字を含めることができます。ただし、コード生成時に、タブ文字は除去され、改行文字は改行文字の直前にエスケープ文字が付加されます。
・SELECT 文の取得列にワイルドカード「*」を使用することはできません。
・「SELECT INTO」句を含む SQL を追加することはできません。ただし SQLJ 実装(後述)の場合、メソッド詳細設定が単一行照会(ただし JOIN により 1:n 関連の親子関係をもつ DTO の取得は除く)のメソッドを生成すると、DAO 生成ツールによって「SELECT」句が「SELECT INTO」句に変換されます。
・ストアード・プロシージャーを使用する場合は、カスタム・メソッドとしてインターフェースにメソッドを追加し、処理を独自に実装する必要があります。
・JOINを含むSQLの場合、取得列にそれぞれの表のレコードを一意に識別するためのキー列(主キー列など)を含める必要があります。詳細は4.2.3 ユーザー定義CRUDメソッドの編集を参照してください。

(例)
SELECT E.**EMPNO**, E.FIRSTNAME, D.**DEPTNO**, D.DEPTNAME FROM EMPLOYEE E, DEPARTMENT D WHERE E.DEPTNO = D.DEPTNO

4.2.3. ユーザー定義CRUDメソッドの編集

『DAO 定義』エディターでは、ユーザー定義 CRUD メソッドが持つ次の属性を変更することが可能です(カスタム・メソッドを除く)。

1. メソッド名
2. SQL 発行時のステートメント種別(java.sql.Statement または java.sql.PreparedStatement)
3. 戻り値/引数の型
4. 発行する SQL
5. SQL 取得列(パラメーター)と DTO フィールドの対応関係

メソッド名は、エディター上の「メソッド一覧」に表示されているメソッド名をダブルクリックすることによって、直接編集することが可能です。**ただし、後述する「メソッド詳細設定」の設定を変更すると、メソッド名も自動的に変更されることに注意してください。**また、自動生成された「単一行照会」、「単一行挿入」、「単一行削除」、「単一行更新」メソッドのメソッド名を変更することはできません。2.SQL 発行時のステートメント種別および 3.戻り値/引数の型は、エディター上の「メソッド詳細設定」セクションで変更することが可能です。4.発行する SQL は「SQL 文」、5.SQL 取得列と DTO フィールドの対応関係は「取得列項目/パラメーター項目」によって変更することが可能です。以降では、2 から 5 までの変更方法の詳細について説明します。

SQL 発行時のステートメント種別の変更

エディター上「メソッド詳細設定」の「ステートメント種別」プルダウンによって、「Statement」または「PreparedStatement」を選択します。デフォルトは「PreparedStatement」であり、**通常は変更する必要はありません。**

メソッドの戻り値/引数の変更

エディター上「メソッド詳細設定」では、「ステートメント種別」の他に次のような設定を変更することができます。各設定とメソッドとの対応関係は後述します。
StatementType のタイプは以下から選択することが可能です。

1. 照会系

StatementType	メソッド名	戻り値	備考
単一行照会(Default)	find(Argument *[1])	DTO	● 1DTO に該当するレコードの照会処理を行う場合に選択する。 ● SQL の照会条件に該当するレコードが存在しない場合、メソッドの戻り値が null となる。
単一行照会(Default:0 件時例外)	select(Argument *[1])	DTO	● 1DTO に該当するレコードの照会処理を行う場合に選択する。 ● SQL の照会条件に該当するレコードが存在しない場合、DTONotFoundException が発生する。
単一行照会	find*SQLID*(Argument *[1])	DTO	● ユーザー定義 CRUD。 ● 1DTO に該当するレコードの照会処理を行う場合に選択する。 ● SQL の照会条件に該当するレコードが存在しない場合、メソッドの戻り値が null となる。 ● SQL が 1:n の関係にある表間の JOIN を含まない場合、照会対象は常に 1 レコードとなる。一方、**SQL が、1:n の関係にある表間の JOIN を含む場合は、複数のレコードが操作対象となる可能性がある。**

StatementType	メソッド名	戻り値	備考
単一行照会(0 件時例外)	select*SQLID*(Argument *[1])	DTO	● ユーザー定義 CRUD。 ● 1DTO に該当するレコードの照会処理を行う場合に選択する。 ● SQL の照会条件に該当するレコードが存在しない場合、DTONotFoundException が発生する。 ● SQL が 1:n の関係にある表間の JOIN を含まない場合、照会対象は常に 1 レコードとなる。一方、**SQL が、1:n の関係にある表間の JOIN を含む場合は、複数のレコードが操作対象となる可能性がある。**
複数行照会	findList*SQLID*(Argument *[1])	List	● ユーザー定義 CRUD。 ● 複数の DTO に該当するレコードの照会処理を行う場合に選択する。 ● メソッドの戻り値が(DTO 型の)List となる。 ● 複数の DTO に該当するレコードの照会処理を行う場合に選択する。**SQL が、1:n の関係にある表間の JOIN を含む場合は、戻り値の List のサイズとレコード数が異なる可能性がある。**
複数行照会(範囲指定)	findList*SQLID*(Argument *[1], int)/ findList*SQLID*(Argument*[1], int, int)	List	● ユーザー定義 CRUD。 ● SQL によって、複数の DTO に該当するレコードの照会処理を行う場合に選択する。 ● メソッドの戻り値が(DTO 型の)List となる。 ● SQL によって、複数の DTO に該当するレコードの照会処理を行う場合に選択する。**SQL が、1:n の関係にある表間の JOIN を含む場合は、戻り値の List のサイズとレコード数が異なる可能性がある。** ● findList*SQLID*(Argument*[1], int) および findList*SQLID*(Argument*[1], int, int) の 2 メソッドが出力される。 ● findList*SQLID*(Argument*[1], int)の場合、取得する行数の最大値を指定することができる。 ● findList*SQLID*(Argument*[1], int, int)の場合、取得する範囲の開始行数と、取得する行数を指定することにより、照会結果の取得範囲を指定することができる。
単一行照会(Map)	find*SQLID*(Argument*[1])	Map	● ユーザー定義 CRUD。 ● SQL によって、1DTO に該当するレコードの照会処理を行う場合に選択する。 ● SQL の照会条件に該当するレコードが存在しない場合、メソッドの戻り値が null となる。 ● 関数や外部結合など、通常の DTO では取得列を格納できないような場合に選択する。 ● 選択により、戻り値が java.util.Map 型となる。
単一行照会(Map：0 件時例外)	select*SQLID*(Argument*[1])	Map	● ユーザー定義 CRUD。 ● SQL によって、1DTO に該当するレコードの照会処理を行う場合に選択する。 ● SQL の照会条件に該当するレコードが存在しない場合、DTONotFoundException が発生する。 ● 関数や外部結合など、通常の DTO では取得列を格納できないような場合に選択する。 ● 選択により、戻り値が java.util.Map 型となる。

StatementType	メソッド名	戻り値	備考
複数行照会(Map)	findList*SQLID*(Argument*[1])	Map List	● ユーザー定義 CRUD。 ● SQL によって、複数の DTO に該当するレコードの照会処理を行う場合に選択する。 ● 関数や外部結合など、通常の DTO では取得列を格納できないような場合に選択する。 ● 選択により、戻り値が java.util.Map 型の List 型となる。
複数行照会(Map:範囲指定)	findList*SQLID*(Argument*[1], int)/ findList*SQLID*(Argument*[1], int, int)	Map List	● ユーザー定義 CRUD。 ● SQL によって、複数の DTO に該当するレコードの照会処理を行う場合に選択する。 ● 関数や外部結合など、通常の DTO では取得列を格納できないような場合に選択する。 ● 選択により、戻り値が java.util.Map 型の List 型となる。 ● findList*SQLID*(Argument*[1], int) および findList*SQLID*(Argument*[1], int, int)の 2 メソッドが出力される。 ● findList*SQLID*(Argument*[1], int)の場合、取得する行数の最大値を指定することができる。 ● findList*SQLID*(Argument*[1], int, int)の場合、取得する範囲の開始行数と、取得する行数を指定することにより、照会結果の取得範囲を指定することができる。
カーソル照会	getCursor*SQLID*(Argument*[1])	Cursor	● ユーザー定義 CRUD。 ● ロックや逐次更新などを行う場合に選択する。 ● 選択により、メソッドの戻り値が Cursor 型となる。 ● **JOIN を含む SQL に使用することはできない。**
カーソル照会(Map)	getMapCursor*SQLID*(Argument*[1], int)	Map Cursor	● ユーザー定義 CRUD。 ● ロックや逐次更新などを行う場合に選択する。 ● 選択により、戻り値が MapCursor となる

表 4-5 照会系メソッド一覧

*[1] Parameter 型、または、primitive ラッパー型が指定可能
V7.1.1 から、DAO メソッドの引数として Parameter クラスの他、primitive ラッパー型(java.lang.Integer など)が使用可能になりました。primitive ラッパー型では、Parameter に相当する部分の引数が可変になります。
primitive ラッパー型の場合、必ずパラメーターが指定されるため、動的パラメーター照会機能を使用することは出来ません(IS NULL 自動変換、IS NOT NULL 自動変換は有効です)。
primitive型のラッパーを使用するには、表 5-1 プロパティー一覧の「argumentType」の項目を参照してください。

単一行照会を選択したメソッドは、SQLJ 実装を選択している場合、DAO 生成ツールにより「SELECT INTO」構文に変換されます。よって、**SQLJ 実装で単一行照会を使用する場合は、「SELECT INTO」構文で使用できる SQL を設定する必要があります。**例えば、UNION を含む SQL は「SELECT INTO」構文ではサポートされないため、使用しないで下さい。「SELECT INTO」構文の詳細は、DB2 のインフォメーション・センター等を参照して下さい。

カーソル照会を選択した場合、メソッド名は「getCursor*SQLID*()」となり、戻り値は Cursor 型となります。SQLJ 実装を選択している場合は、カーソルによる更新、削除時に使用する SQL-ID を指定する必要があります。また、カーソルによる更新、削除時に使用する SQL では、WHERE CURRENT OF 句がない場合、DAO 生成ツールにより WHERE CURRENT OF 句が補完されます。

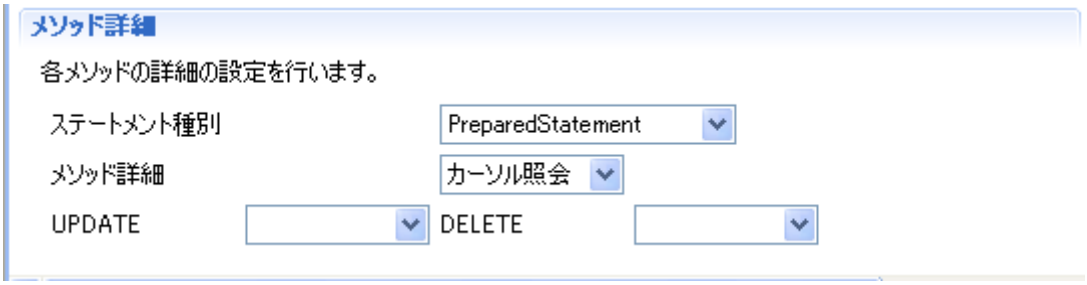


図 4-14 カーソル照会

（注意）
取得列にLOB（※）を含むSQLを使用する場合、複数行照会(範囲指定)のfindList(Argument*[1], int, int)を使用することはできません。複数行照会(範囲指定)のfindList(Argument*[1], int)を使用してください。
※LOB Large Object の略。サイズの大きなデータを扱うための列データ型。

2. 更新系
また、更新系のメソッド名は次のように設定されます。

選択画面	メソッド名	戻り値の型	備考
挿入処理			
単一行挿入(Default)	insert(DTO)	void	• DTO 指定更新 • 該当行がない場合は、例外発生
単一行挿入	insert*SQLID*(DTO)	void	• ユーザー定義 CRUD • DTO 指定更新 • 該当行がない場合は、例外発生
バッチ挿入	insert*SQLID*(List)	void	• ユーザー定義 CRUD • DTO(List)指定更新 • 該当行がない場合は、例外発生
複数行挿入	insert*SQLID*(Argument*[1])	int	• ユーザー定義 CRUD • Parameter 指定更新 • 該当行がない場合は、0 を返す
更新処理			
単一行更新(Default)	update(DTO)	void	• DTO 指定更新 • 該当行がない場合は、例外発生
単一行更新	update*SQLID*(DTO)	void	• ユーザー定義 CRUD • DTO 指定更新 • 該当行がない場合は、例外発生
バッチ更新	update*SQLID*(List)	void	• ユーザー定義 CRUD • DTO(List)指定更新 • 該当行がない場合は、例外発生
複数行更新	update*SQLID*(Argument*[1])	int	• ユーザー定義 CRUD • Parameter 指定更新 • 該当行がない場合は、0 を返す
カーソル更新	update(DTO)	void	• ユーザー定義 CRUD • カーソル・クラスに出力される • 編集・削除不可
削除処理			
単一行削除(Default)	delete(DTO)	void	• DTO 指定更新 • 該当行がない場合は、例外発生
単一行削除	delete*SQLID*(DTO)	void	• ユーザー定義 CRUD • DTO 指定更新 • 該当行がない場合は、例外発生
バッチ削除	delete*SQLID*(DTO)	void	• ユーザー定義 CRUD • DTO(List)指定更新 • 該当行がない場合は、例外発生

選択画面	メソッド名	戻り値の型	備考
複数行削除	delete*SQLID*(Argument*[1])	int	• ユーザー定義 CRUD • Parameter 指定更新 • 該当行がない場合は、0 を返す
カーソル削除	delete()	void	• ユーザー定義 CRUD • カーソル・クラスに出力される • 編集・削除不可

表 4-6 更新系メソッド一覧

*[1] Parameter 型、または、primitive ラッパー型
*[1] V7.1.1 から、DAOメソッドの引数としてParameterの代わりにprimitive型のラッパーを使用できるようになり、この場合はParameterに相当する部分の引数が可変になります。primitive型のラッパーを使用するには、表 5-1 プロパティー一覧の「argumentType」の項目を参照してください。

（注意）
メソッドタイプがカーソル更新/カーソル削除のメソッドは、編集および削除することができません。カーソル照会の紐付けを解除してから、編集・削除を行なってください。

発行する SQL の変更
エディター上の「編集」ボタンを押下し、『SQL の編集』ダイアログ上で発行する SQL を変更することができます。

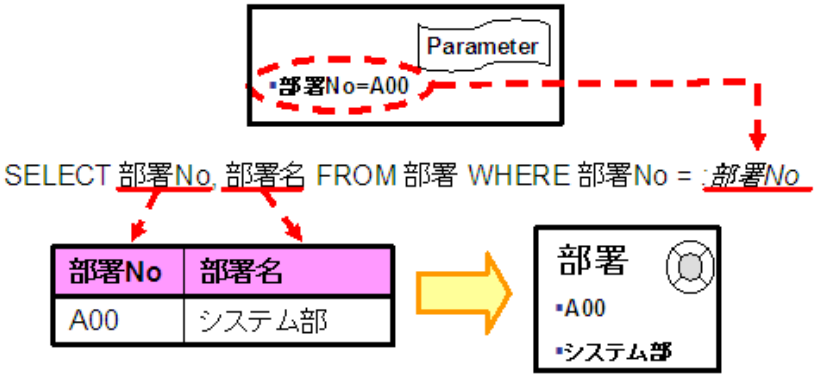
SQL 取得列(パラメーター)と DTO フィールドの対応関係の変更
メソッドのパターンによって、SQL 取得列(パラメーター)と DTO フィールドの対応関係は異なります。

パターン	種別/メソッド	引数/戻り値	JOIN 有無
1	照会系/select, find, findList カーソル照会/getCursor	Argument*[1]/DTO(List) Argument*[1]/Cursor	無し
2	照会系/select, find, findList カーソル照会(Map)/getMapCursor	Argument*[1]/DTO(List) Argument*[1]/MapCursor	有り
3	照会系/select, find, findList	Argument*[1]/Map(List)	N/A
4	更新系/insert, update, delete	DTO(List)/-	N/A
5	更新系/insert, update, delete	Argument*[1]/-	N/A

表 4-7 メソッドのパターン一覧

*[1] Parameter 型、または、primitive ラッパー型

パターン 1: Argument に設定した値が、SQL のパラメーターとして設定されます。また、SQL の取得列が DTO のフィールドに設定されます。エディター上の「パラメーター項目」では、プルダウンによってパラメーター値の型を選択することができます。エディター上の「取得列項目」では、『DTO へのマッピング』ダイアログによって SQL の取得列と DTO のフィールドの対応関係を設定することができます。なお、SQL の取得列名と DTO のフィールドに対応する列名が等しい場合は、DAO 生成ツールによって対応関係が自動的に設定されます。



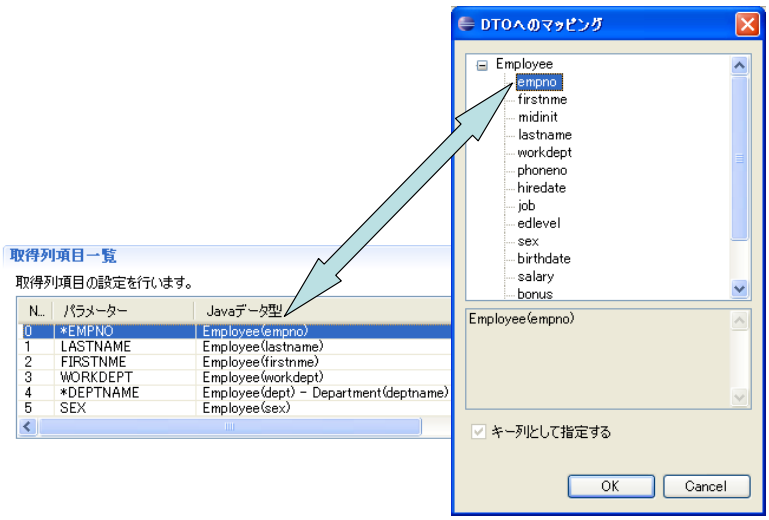
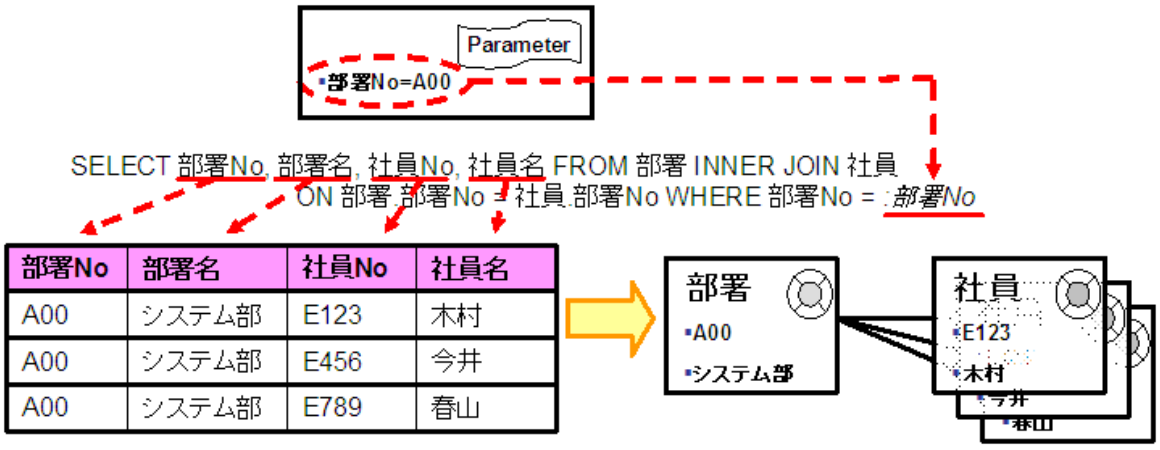


図 4-15 SQL 取得列と DTO フィールドの対応関係(パターン 1)

パターン 2: Argument に設定した値が、SQL のパラメーターとして設定されます。また、SQL の取得列が複数の DTO のフィールドに設定されます。パラメーター値の型選択、および、SQL 取得列と DTO フィールドの対応関係設定方法は、パターン 1 と同様です。パターン 1 と異なるのは、『DTO へのマッピング』ダイアログ上での操作です。パターン 2 では、各 DTO に親子関係があるため※、ダイアログ上で親 DTO が持つフィールドのツリーを展開し、子 DTO のフィールドに対して取得列を設定してください。なお、パターン 2 で対応関係を正しく設定するためには、**JOIN する各表に一意キーが存在し、かつ、SQL の取得列にそれぞれの一意キーを含める必要があります。**例えば JOIN する表間に 1:n の関係があり、かつ、複数の親 DTO(List)を取得するような場合、親表の一意キーが SQL の取得列に含まれていないと、複数のレコードから複数の親 DTO を判別して抽出することができません。
※DTOの親子関係は『DTO定義』エディター上で事前に設定しておいてください。設定方法の詳細は「3.3.2 関連項目」をご覧ください。



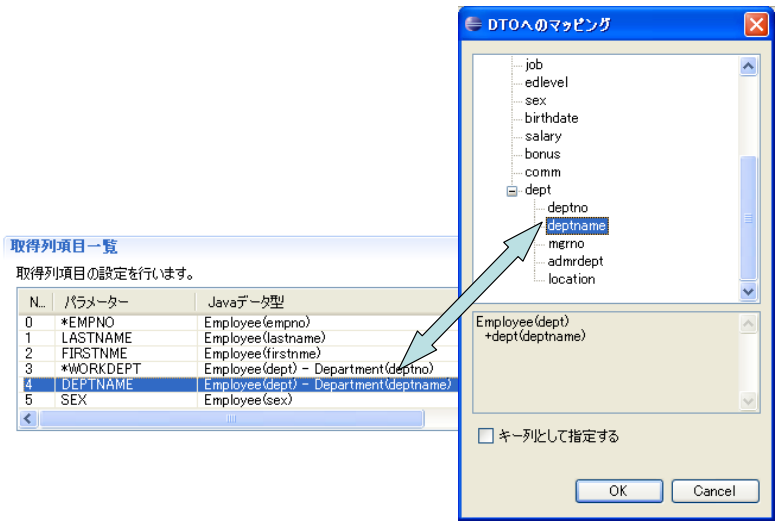


図 4-16 SQL 取得列と DTO フィールドの対応関係(パターン 2)

パターン 3: Argument に設定した値が、SQL のパラメーターとして設定されます。また、SQL の取得列名をキー、取得列を値とした組が、Map の 1 要素として設定されます。エディター上の「パラメーター項目」および「取得列項目」では、プルダウンによってパラメーターおよび取得列の型を選択することができます。

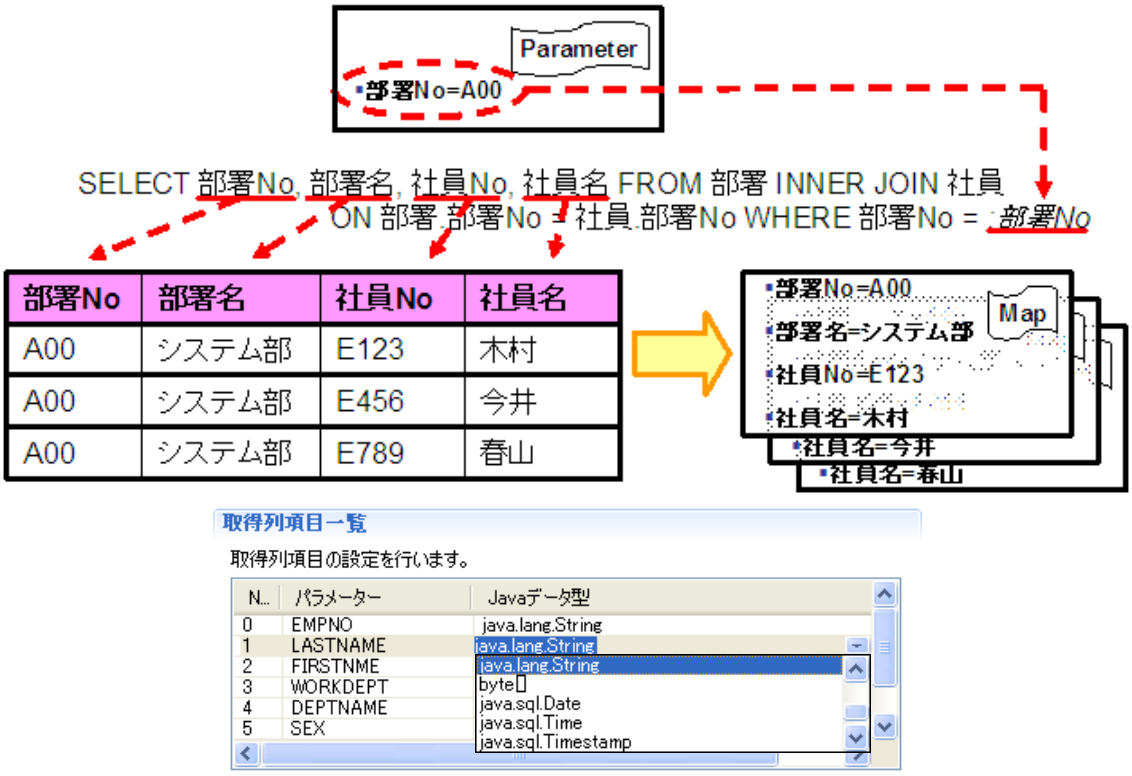


図 4-17 SQL 取得列と Map の対応関係(パターン 3)

パターン 4:DTO のフィールドが、SQL のパラメーターとして設定されます。エディター上の「取得列項目」では、『DTO へのマッピング』ダイアログによって SQL のパラメーターと DTO のフィールドの対応関係を設定することができます。

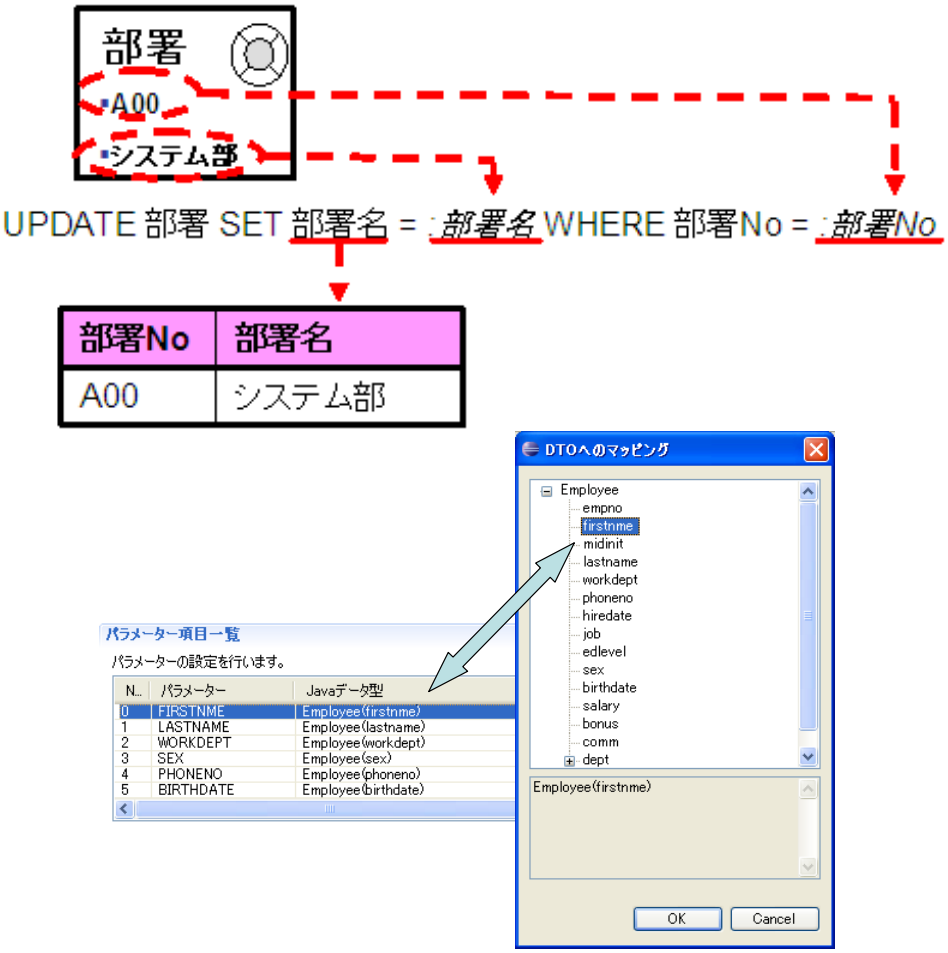


図 4-18 SQL パラメーターと DTO フィールドの対応関係(パターン 4)

パターン 5: Argument に設定した値が、SQL のパラメーターとして設定されます。エディター上の「パラメーター項目」では、プルダウンによってパラメーターの型を選択することができます。

図 4-19 SQL パラメーターと Parameter の対応関係(パターン 5)

5. コード生成

DAO 生成ツールでは、入力された「DAO 定義情報」/「DTO 定義情報」をもとに DAO クラス、DTO クラス、および、その他のクラスの生成を行います。この章では、コードの生成方法について説明します。

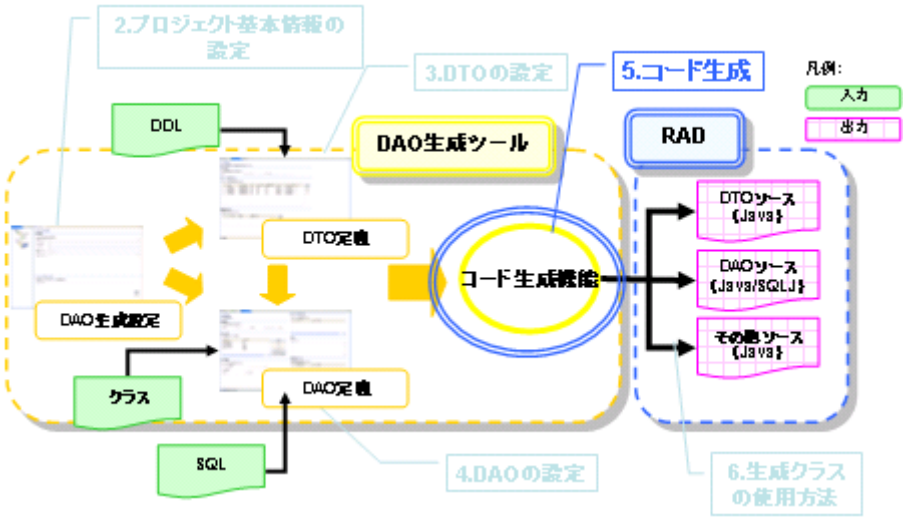


図 5-1 コード生成

5.1.1. コードの生成

コードの生成方法には、プロバイダーや出力するコードを選択してコードを生成する「生成」と、初回「生成」時に保存された設定でデ全てのコードを生成する「全て生成」の 2 種類があります。

生成

プロバイダーの選択、出力するコードの選択を行い、それに従いコードを生成します。

手順は、以下の通りです。

1. 「アウトライン」ビューから「DB」を右クリックし、「生成」を選択します。すると『生成ダイアログ』が起動し、『プロバイダー設定』ウィザードが表示されます。

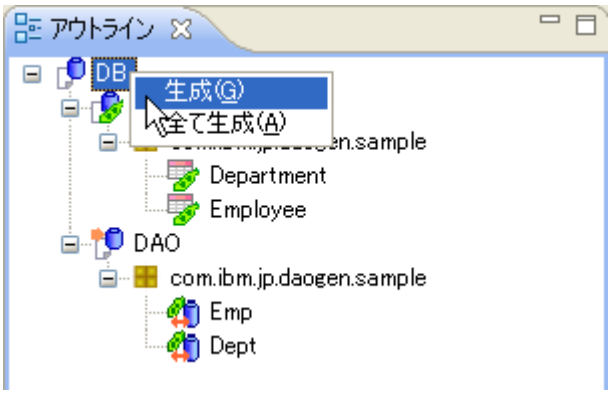


図 5-2 コード生成

2. 『プロバイダー設定ウィザード』では、出力するクラスに関する設定を行います。

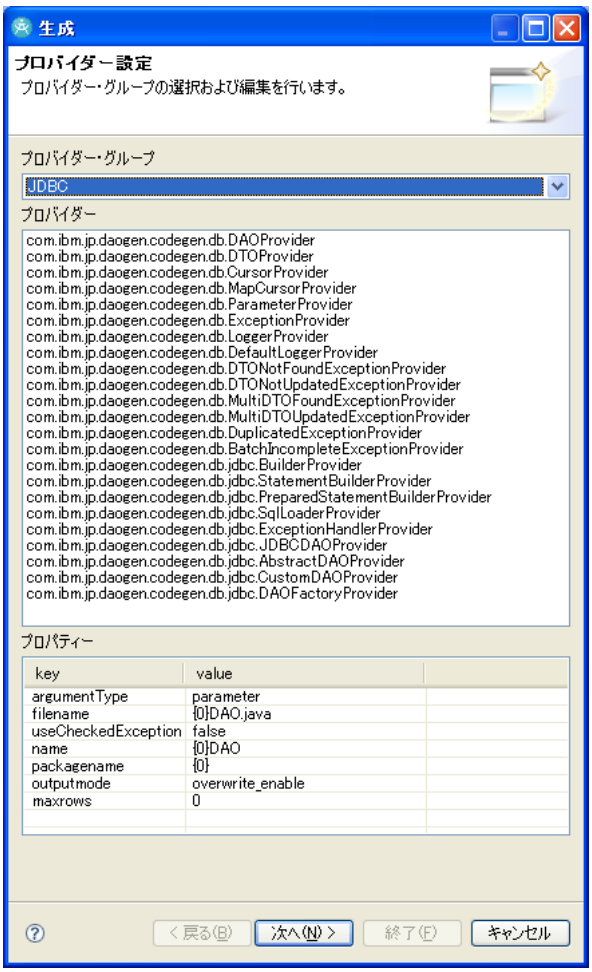


図 5-3 プロバイダー・グループの選択

ウィザード上に表示される『プロバイダー・グループ』では、実装を選択します。Base 版は「JDBC」または「SQLJ」、Express 版は「Spring(JDBC)」または「Spring(SQLJ)」から選択することができます。

『プロバイダー』は、コード生成の際に使用するクラス一覧です。各プロバイダーに関して、プロパティーを設定することができます。設定を変更したいプロバイダーを『プロバイダー』から選択し、『プロパティー』上で編集してください。『プロパティー』に表示される主なキーの意味は、次のとおりです。

キー	説明
name	クラス名を指定。{0}は *DAOName*、*DTOName* によって置換される。
filename	ファイル名を指定。**拡張子を除いたファイル名とクラス名は同一である必要がある。**
packagename	クラスが所属するパッケージ名を指定。{0}は『新規 DAO 生成設定作成』ウィザードで指定したパッケージ名によって置換される。Parameter クラスや DataAccessException クラスなどの共通クラスは同一パッケージを設定することが可能。
outputmode	output_enable :コード生成を実行、すでに同名のファイルが存在する場合は生成しない overwrite_enable:コード生成を実行、すでに同名のファイルが存在する場合上書きする output_disenable:コード生成を行わない
configurationFileName	定義ファイルを指定。
superClassName	継承する親クラスを指定。
nullable	(DTOProvider のみ) false に設定した場合、DTO に設定した null 値は全て初期値へ変換される。

キー	説明
accessorNameStrict	(DTOProvider, AbstractDAOProvider) DTO の getter/setter メソッド名を JavaBean 仕様に準拠させるかどうかを指定。2 つの Provider の設定は同一にする必要がある。 true : JavaBean 仕様に準拠 (デフォルト) false : フィールド名の先頭 1 文字のみを大文字に変換
Impl_type	(DefaultLoggerProviderのみ) 値(Default / wacs / commons-logging)に応じて、ログ出力処理の実装クラスを切り替える。詳細は「6.2.4 Loggerクラス」を参照。
updateColumns	(「SQLJ」、「Spring(SQLJ)」実装の AbstractDAOProvider のみ) 更新可能 Iterator に対して updateColumns オプションを付与するか否かを true/false にて設定。
retrievalArgument	(AbstractDAOProvider のみ) 生成コード内の結果セット取得に使用する API を切り替える。 positioned : 列番号指定(デフォルト) name : 列名指定
argumentType	(DAOProvider, AbstractDAOProvider)DAO メソッドの引数(Parameter クラス/ primitive ラッパー型)を切り替える。2 つの Provider の設定は同一にする必要がある。 parameter : Parameter クラス(デフォルト) wrapper : primitive ラッパー型
trim	(AbstractDAOProvider のみ) 結果セットの CHAR/CHARACTER 型の文字列の後部を右トリムして取得するかどうかを指定。 true :右トリムを行う(デフォルト) false :右トリムを行わない
optimizeForSelectIntoStatement	(「SQLJ」、「Spring(SQLJ)」実装のAbstractDAOProviderのみ) SELECT INTO構文に変換されるSQLに対して、FOR READ ONLY句またはFOR FETCH ONLY句またはFOR UPDATE (OF カラム名)句を削除するかどうかをtrue/falseにて設定。SELECT INTO構文への変換条件については、4.2.2 使用可能なSQLのその他を参照。SELECT INTO ... FOR READ ONLY（FOR FETCH ONLY句および、FOR UPDATE句も同様）は、DB2 for LUWではサポートされる構文であるが、DB2 for z/OSでは構文エラーになるため、上記 3 つの句のいずれかがSQL上に記述されている場合、その記述部分を削除してエラーを回避する。（デフォルト値：false）

表 5-1 プロパティー一覧

3. 『プロバイダー定義』ウィザード上で『次へ』ボタンを押下すると、『生成ソースコード選択』ウィザードが表示されます。コードを生成するクラスにチェックを入れ、『終了』ボタンを押下すると、コードが生成されます（DAO・DTO以外のコードは、コード生成時に自動的に生成されます。）。なお、「出力内容が同一の場合は上書きしない」を選択した場合、既存のソースコードと生成するソースコードを比較し、相異のある場合だけ内容を上書きします。

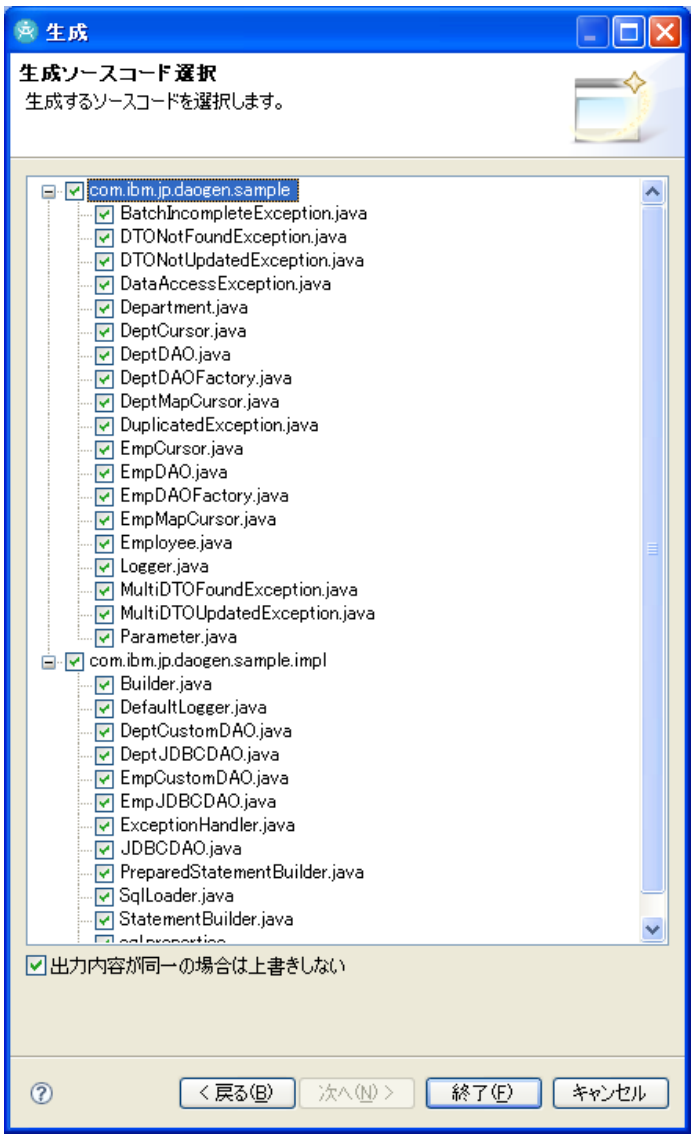


図 5-4 生成ソースコード選択

全て生成

「生成」にて設定した設定に従い(※)、すべてのコードを生成します。

(※)「生成」前に選択した場合例外が発生します。初回生成時は、まず「生成」を選択し、プロバイダー・グループなどを設定する必要があります。

手順は以下の通りです。

1. 「アウトライン」から「DB」を右クリックし、「全て生成」を選択します。

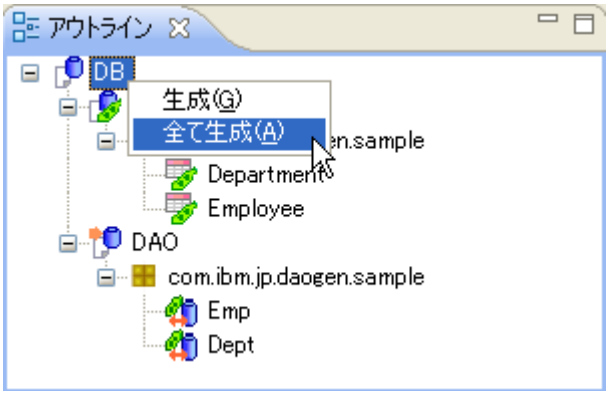


図 5-5 全てのコードを生成

2. ソースフォルダに、全てのソースコードが出力されます。

5.1.2. SQL定義ファイルについて

SQL 定義ファイルの扱いは、コード生成ウィザードで選択したプロバイダー・グループにより異なります。

① 「JDBC」/「Spring(JDBC)」を選択した場合：「SqlLoaderProvider」の「sqlPackageName」に指定したフォルダーへ出力されます。デフォルトでは各 DAO 実装クラスと同一ディレクトリー内に出力されます。

② 「SQLJ」/「Spring(SQLJ)」を選択した場合：SQL 文は DAO クラス内に出力されます、SQL 定義ファイルは出力されません。

6. 生成クラスの使用方法

この章では、DAO 生成ツールによって生成されたクラス(DAO・DTO クラスなど)の使用方法について説明します。

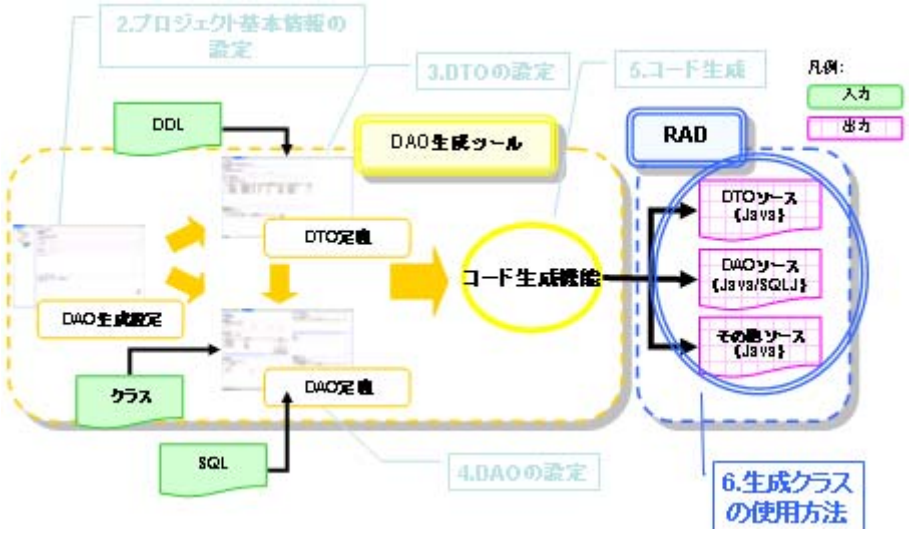


図 6-1 生成クラスの使用方法

6.1. 生成クラス概要

1 つの DAO・DTO 定義情報(DAO 定義/DTO 定義)につき、次のようなクラス群が生成されます。なお、『新規 DAO 生成設定ファイルの作成』ウィザードで選択したプロバイダー・グループ(JDBC/SQLJ/Spring JDBC/Spring SQLJ)により、生成されるクラスは一部異なります。また、内部処理用のクラスは説明を省略しています。

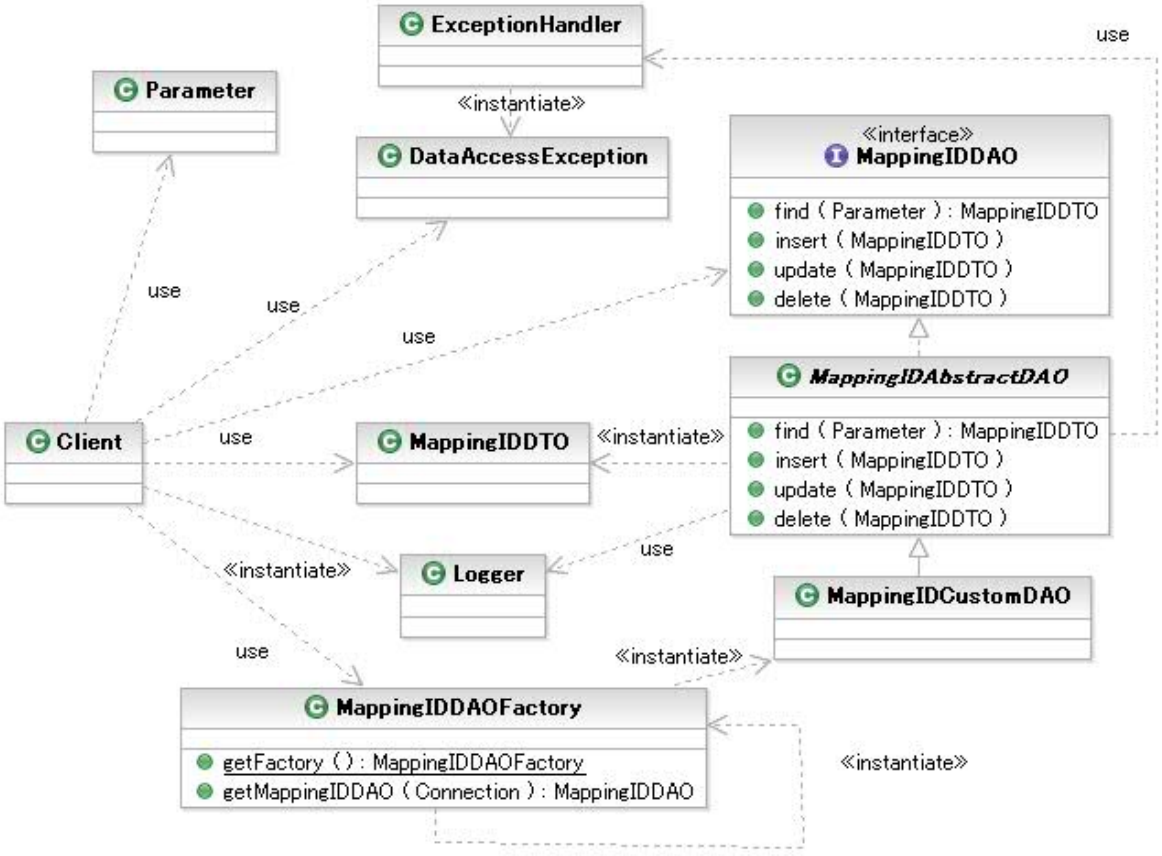


図 6-2 生成クラスの概要(Client クラスは、DAO を使用する任意のクラス)

パッケージ (デフォルト)	クラス名	概要	自動生成
packagename	*DTOName*	1 表の照会結果に対応する DTO クラスです。	毎回
	DAONameDAO	DAO のインターフェースです。照会時には複数の表にアクセスする可能性があります。	毎回
	DAONameDAOFactory	DAO を生成するファクトリー・クラスです。デフォルトでは *DAONameCustomDAO* クラスをインスタンス化します。	1 回
	DataAccessException	DAO から throw される例外クラスです。実際には、このクラスのサブクラスが throw される場合があります。	1 回
	Parameter	DAO 呼び出し時に使用するパラメーター保持用クラスです。	1 回
	Logger	ロギング用インターフェースです。DAO 呼び出し時に使用します。実際には DefaultLogger クラスなどの実装クラスを用意する必要があります。	1 回
packagename.impl	*DAONameAbstract*DAO	DAO の抽象クラスです。実装種別によって内容が異なります。*Abstract* 部分は、「JDBC」、「SQLJ」、「SpringJDBC」、「SpringSQLJ」のいずれかに置換されます。	毎回
	DAONameCustomDAO	DAO の抽象クラスのサブクラスです。抽象クラスのメソッドをオーバーライドすることができます。	1 回
	ExceptionHandler	DAO 内で発生する例外を制御するためのクラスです。	1 回

表 6-1 各クラスの概要

上記クラスを使用した、基本的なプログラムの流れは次のとおりです。

```
// Connection 取得(任意の方法)
Connection conn = getConnection(args);

// Factory 生成
EmployeeDAOFactory factory = EmployeeDAOFactory.getFactory();

// DAO インスタンス取得
EmployeeDAO dao = factory.getEmployeeDAO(conn);

// 単一行挿入メソッドによる挿入処理実行
Employee dto = new EmployeeDTO();
dto.setEmpno("000010");
dto.setFirstnme("DAOTARO");
dao.insert(dto);

// 単一行照会メソッドによる照会処理実行
Parameter param = new Parameter();
param.setString("EMPNO", "000010");
dto = dao.find(param);

// 単一行更新メソッドによる更新処理実行
dto.setFirstnme("DTOTARO");
dao.update(dto);

// 単一行削除メソッドによる削除処理実行
dao.delete(dto);
```

6.2. APIの詳細

ここでは、DAO 生成ツールが生成するクラスのうち、仕様が比較的複雑な「DAO クラス(*DAONameDAO*/*DAONameAbstractDAO*/*DAONameCustomDAO*)」、「ExceptionHandler クラス」、「Logger クラス」について説明します。

6.2.1. DAOクラス

DAO クラスには表に対する CRUD 操作(INSERT、SELECT、UPDATE、DELETE)を行うメソッドが存在します。各メソッドのうち、「カスタム・メソッド」以外の実装は、全て *DAONameAbstract*DAO クラスに自動生成されます。自動生成されるメソッドの詳細は次のとおりです。「カスタム・メソッド」の詳細は後述します。

機能	該当する API	説明
DTO 取得	・find(Argument*[1]) : *DTOName* ・find*SQLID*(Argument*[1]) : *DTOName* ・select(Argument*[1]) : *DTOName* ・select*SQLID*(Argument*[1]) : *DTOName*	・1 つの DTO インスタンスを取得するための API(ただし、JOIN を含む場合は他の複数の DTO インスタンスを含む)。 ・(Parameter 型の場合) Parameter クラスには、実行する SQL で使用されるホスト変数名をキーとし、値を設定する(ホスト変数名は Javadoc 上で確認することができる)。 ・(primitive ラッパー型の場合) DAO の引数に、実行する SQL で使用するパラメーターを設定する ・該当する DTO インスタンスが無い場合、「find」から始まるメソッドと「select」から始まるメソッドでは挙動が異なる。「find」の場合は null が戻る。一方、「select」の場合は DTONotFoundException が発生する。また、いずれのメソッドも、DTO インスタンスが 2 つ以上該当した場合は MultiDTOFoundException が発生する。
コード例		
Parameter 型の場合 Parameter param = new Parameter(); param.setString("EMPNO", "000010"); EmployeeDTO dto = dao.**find(param)**; **primitive ラッパー型の場合** EmployeeDTO dto = dao.**find(“000010”);**		
機能	**該当する API**	**説明**
複数 DTO 取得	・findList*SQLID*(Argument*[1]) : List ・findList*SQLID*(Argument*[1], int) : List ・ findList*SQLID*(Argument*[1], int, int) : List	・複数の DTO インスタンスを取得するための API。 ・Parameter クラスには、実行する SQL で使用されるホスト変数名をキーとし、値を設定する(ホスト変数名は Javadoc 上で確認することができる)。 ・(primitive ラッパー型の場合) DAO の引数に、実行する SQL で使用するパラメーターを設定する ・int 型の引数にはフェッチ範囲(0～)を指定する。ただし、BLOB/CLOB 列を含む照会処理ではフェッチ開始位置を指定することができない。また、フェッチ開始位置を指定した場合はスクロール可能カーソル(ResultSet.TYPE_SCROLL_INSENSITIVE)となる。 ・List の各要素は DTO インスタンスである。 ・該当する DTO インスタンスの無い場合は、長さ 0 の List が戻る。
コード例		

機能	該当する API	説明

Parameter 型の場合
Parameter param = new Parameter();
param.setString("LASTNAME", "HAAS”);
param.setString(“SEX", "F");
List dtoList = dao.**findSQLID1(param)**;
for (int i = 0; i < dtoList.size(); i++) {
EmployeeDTO dto = (EmployeeDTO) dtoList.get(i);
・・・
}
 	
primitive ラッパー型の場合
String lastname = “HAAS”;
String sex = “F”;
List dtoList = dao.**findSQLID1(lastname, sex);**
・・・(以降は Parameter 型の場合と同様)		

機能	**該当する API**	**説明**
Map DTO 取得	・find*SQLID*(Argument*[1]) : Map ・select*SQLID*(Argument*[1]) : Map ・findList*SQLID*(Argument*[1]) : List ・findList*SQLID*(Argument*[1], int) : List ・ findList*SQLID*(Argument*[1], int, int) : List	・DTO インスタンスの代わりに、取得列名をキー、取得列の値を値とした Map インスタンスが戻る。 ・その他の挙動は上述したメソッドと同様。
コード例		
Parameter 型の場合 Parameter param = new Parameter(); param.setString("EMPNO", "000010"); Map map = dao.**findSQLID1(param)**; **primitive ラッパー型の場合** String empno = “000010” Map map = dao.**findSQLID1(empno)**;		
機能	**該当する API**	**説明**
カーソル取得	・getCursor*SQLID*(Argument*[1]) : *SQLID*Cursor ・getMapCursor*SQLID* (Argument*[1]) : *SQLID*Cursor	・*SQLID*Cursor インスタンスを取得するための API。 ・取得前に、必ず Connection # setAutoCommit(false)を呼び出し、自動コミットを OFF としておく。 ・*SQLID*Cursor クラスには「next()」、「get()」、「update()」、「delete()」の各メソッドが存在し、それらを利用してレコード単位での更新処理などを行う。 ・使用後は、*SQLID*Cursor # close()および Connection # commit()メソッドを呼ぶ。
コード例		

機能	該当する API	説明

conn.setAutoCommit(false);
*SQLID*Cursor cursor = **dao.getCursorSQL1(null)**;
 	
while (cursor.**next()**) {
　EmployeeDTO dto = cursor.**get()**;
　if (*条件 1)* {
　　dto.setFirstnme("DTOTARO");
　　cursor.**update(dto)**;
　}
 	
　if (*条件 2*) {
　　cursor.**delete()**;
　}
}
・・・
conn.commit();
・・・
} finally {
・・・
　　cursor.**close()**;
・・・
}		

機能	該当する API	説明
DTO 指定更新	・insert(*DTOName*) ・insert*SQLID*(*DTOName*) ・update(*DTOName*) ・update*SQLID*(*DTOName*) ・delete(*DTOName*) ・delete*SQLID*(*DTOName*)	・1 つの DTO インスタンスを更新するための API。「insert」、「update」、「delete」の各メソッド内部では、INSERT 文、UPDATE 文、DELETE 文を発行する。 **・更新する DTO インスタンスは常に引数に指定した DTO インスタンス 1 つであり、引数の DTO インスタンスが持つ他の（子供の）DTO インスタンスは更新対象とはならない。** ・該当する DTO インスタンスが無い場合、DTONotUpdatedException が発生する。また、DTO インスタンスが 2 つ以上該当した場合は MultiDTOUpdatedException が発生する。
コード例		

Employee dto = new EmployeeDTO();
dto.setEmpno("000010");
dto.setFirstnme("DAOTARO");
dao.insert(dto);
 	
dao.update(dto);
 	
dao.delete(dto);		

機能	該当する API	説明

機能	該当する API	説明
バッチ更新	・insert*SQLID*(List) ・update*SQLID*(List) ・delete*SQLID*(List)	・複数の DTO インスタンスを更新するための API。「insert」、「update」、「delete」の各メソッド内部では、INSERT 文、UPDATE 文、DELETE 文を発行する。 **・更新する DTO インスタンスは常に引数に指定した DTO インスタンスであり、引数の DTO インスタンスが持つ他の(子供の)DTO インスタンスは更新対象とはならない。** ・いずれかの更新が失敗した場合、全ての処理終了後に BatchIncompleteException が発生する。それぞれの例外は BatchIncompleteException # getExceptions()にて取得することができる。 **・Oracle データベース、かつ、PreparedStatement を使用した場合、バッチ内の 1SQL に対する 0 件更新または複数件更新を検出することができない。**

コード例

```
Employee dto = new EmployeeDTO();
dto.setEmpno("000010");
dto.setFirstnme("DAOTARO");

List list = new ArrayList();
list.add(dto);
list.add(dto);
list.add(dto);

dao.insertSQL1(list);

dao.updateSQL1(list);

dao.deleteSQL1(list);
```

機能	該当する API	説明
Parameter 指定更新	・insert(Argument*[1]) : int ・insert*SQLID*(Argument*[1]) : int ・update(Argument*[1]) : int ・update*SQLID*(Argument*[1]) : int ・delete(Argument*[1]) : int ・delete*SQLID*(Argument*[1]) : int	・複数の DTO インスタンスを更新するための API。「insert」、「update」、「delete」の各メソッド内部では、INSERT 文、UPDATE 文、DELETE 文を発行する。 ・Parameter クラスには、実行する SQL で使用されるホスト変数名をキーとし、値を設定する(ホスト変数名は Javadoc 上で確認することができる)。 ・戻り値は更新した DTO の数(更新行数)である。

コード例

機能	該当する API	説明
Parameter 型の場合 Parameter param = new Parameter(); param.setString(“EMPNO”, “000010”); param.setString(“FIRSTNME”, “DTOTARO”); int result = **dao.insertSQL1(param)**; param = new Parameter(); param.setString(“FIRSTNME”, “DTOTARO”); result = **dao.updateSQL1(param)**; param = new Parameter(); param.setString(“EMPNO”, “000010”); result = **dao.deleteSQL1(param)**; **primitive ラッパー型の場合** String empno = “000010”; String firstname = “DTOTARO”); int result = **dao.insertSQL1(empno, firstname)**; result = **dao.updateSQL1(firstname)**; result = **dao.deleteSQL1(empno)**;		

表 6-2 自動生成されるメソッド一覧

DAO クラスには、上記以外に「カスタム・メソッド」が存在します。「カスタム・メソッド」とは、*DAOName*DAO インターフェースに宣言されたメソッドのうち、*DAONameAbstract*DAO クラスによって実装されていないメソッドのことです。「カスタム・メソッド」は *DAONameAbstract*DAO クラスのサブクラスである、*DAOName*CustomDAO クラスに実装する必要があります。

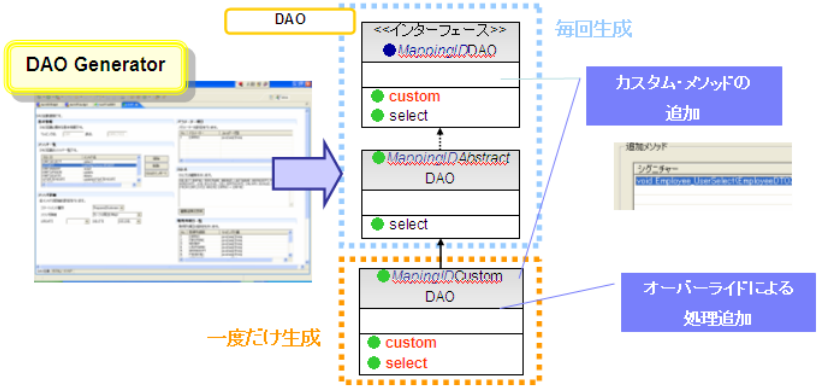


図 6-3 *DAOName*CustomDAO クラスの位置づけ

*DAOName*CustomDAO クラスの内容は、意図的に削除しない限り、DAO 生成ツールによって更新されることはありません。したがって、*DAOName*DAO インターフェースのメソッドを実装するだけでなく、*DAONameAbstract*DAO クラスのメソッドをオーバーライドし、独自の処理を追加することも可能です。

6.2.2. Parameterクラス

Parameter クラスは SQL 文内のパラメーターに対する値を DAO へ渡すためのクラスです。SQL 文内のホスト変数名をキーとして、その値を指定します。指定する値が String 型の場合は次のように値を指定します。値が String 型以外の場合は型名に応じた setXXX メソッドを使用してください(Date 型の場合：setDate(String, Date)など)。

```
// SQL 文
// SELECT EMPNO, FIRSTNME FROM EMPLOYEE WHERE EMPNO = :EMPNO

Parameter param = new Parameter();
param.setString("EMPNO", "000010");
```

```
EmployeeDTO dto = dao.find(param);

// 変換後の SQL (PreparedStatement の場合)
// SELECT EMPNO, FIRSTNME FROM EMPLOYEE WHERE EMPNO = ?
```

動的パラメーター照会機能を使用した場合は、パラメーターの設定有無によって照会条件が動的に変化します。詳細は「4.2.2使用可能なSQL」を参照してください。

また、replace(String, String)メソッドによりSQL 文自体を置換することも可能です(「SQLJ」実装を除く)。ただし、このメソッドを使用する場合は SQL インジェクションに対する処理を呼び出し側で必ず行う必要があります。**なお、replace メソッドでは動的パラメーター照会機能を使用することはできません。**

```
// SQL 文
// SELECT EMPNO, FIRSTNME FROM EMPLOYEE WHERE EMPNO = :EMPNO ORDER BY :ORDER

Parameter param = new Parameter();
param.setString("EMPNO", "000010");
param.replace("ORDER", "EMPNO");
EmployeeDTO dto = dao.find(param);

// 変換後の SQL (PreparedStatement の場合)
// SELECT EMPNO, FIRSTNME FROM EMPLOYEE WHERE EMPNO = ? ORDER BY EMPNO
```

その他、setQueryTimeout(int)メソッドにより照会時の最大待機時間を指定したり、CURRENT_DATE 定数(java.sql.Date 型)/CURRENT_TIMESTAMP 定数(java.sql.Timestamp 型)により、SQL の CURRENT_TIMESTAMP 関数を使用することが可能です(「SQLJ」実装を除く)。

```
// SQL 文
// UPDATE EMPLOYEE SET HIREDATE = :HIREDATE, PHONENO = :PHONENO

EmployeeDTO dto = //
dto.setHiredate(Parameter.CURRENT_DATE);
dto.setPhoneno("XXXX");
dao.update(dto);

// 変換後の SQL (PreparedStatement の場合)
// UPDATE EMPLOYEE SET HIREDATE = CURRENT_DATE, PHONENO = ?
```

6.2.3. ExceptionHandlerクラス

ExceptionHandler クラスには次の 5 つのメソッドが存在します。

- handleQueryException(Exception e, int state) throws DataAccessException
- handleQueryException(Exception e, int state, Object[] param) throws DataAccessException
- handleUpdateException(Exception e, int state) throws DataAccessException
- handleUpdateException(Exception e, int state, Object[] param) throws DataAccessException
- handleBatchException(Exception e, int[] results, List dtoList) throws DataAccessException

handleQueryException は、DAO の照会系メソッド内部で例外が発生した際に呼び出されます。
handleUpdateException は、DAO の更新系メソッド内部で例外が発生した際に呼び出されます。
handleQueryException、handleUpdateException は 2 種類のメソッドが存在しますが、そのうち Object 配列を引数に持つメソッドは、根本例外が存在しないケース(単一行照会での 0 件照会または複数件照会、単一行更新/削除での 0 件更新または複数件更新)で使用されます。
handleBatchException は、DAO のバッチ更新メソッド内部で例外が発生した際に呼び出されます。引数 e には、SQLException、IOException をはじめ、他の例外クラスが渡される可能性があります。また、引数 state はメソッドが呼び出された時点の状態を表します。state には次の定数値のうち、「NOTFOUND」、「MULTIFOUND」、「NOTUPDATED」、「MULTIUPDATED」、「UNKNOWN」のいずれかが渡されます。

状態	説明
NORMAL	正常に処理が終了した場合。
NOTFOUND	DTO 取得照会(「select」から始まるメソッド)実行時に、該当する DTO が存在しなかった場合。
MULTIFOUND	DTO 取得照会(「select」から始まるメソッド)実行時に、該当する DTO が複数存在した場合。

NOTUPDATED	DTO 指定更新実行時に、該当する DTO が存在しなかった場合。
MULTIUPDATED	DTO 指定更新実行時に、該当する DTO が複数存在した場合。
UNKNOWN	上記以外に例外が発生した場合。例えば、BLOB 取得時のストリーム入出力例外など。

表 6-3 ExceptionHandler が扱う状態一覧

引数 param には、例外発生元の情報が設定されます。param には以下の情報が設定されます。

要素番号	設定値
0	例外が発生した DAO のクラス名
1	例外が発生したメソッド名
2	SQL ID
3	SQL
4	handleQueryException の場合は Parameter handleUpdateException の場合は DTO

表 6-4 ExceptionHandler に渡される例外発生元の情報一覧

results にはバッチ登録された SQL の更新結果、dtoList にはバッチ更新メソッドの引数に指定した DTO のリストが渡されます。
独自の ExceptionHandler を DAO に設定することも可能です。*DAOName*DAOFactory #get*DAOName*DAO(Connection, ExceptionHandler)メソッドを使用して DAO インスタンスを取得し、第二引数に独自の ExceptionHandler クラスを指定してください。

```
// ExceptionHandler クラスの宣言(匿名内部クラス)
ExceptionHandler handler = new ExceptionHandler() {
   public void handleQueryException(Exception e, int state) throws DataAccessException {
      // 独自例外処理
   }
}

// Connection 取得(任意の方法)
Connection conn = getConnection(args);

// Factory 生成
EmployeeDAOFactory factory = EmployeeDAOFactory.getFactory();

// DAO インスタンス取得
EmployeeDAO dao = factory.getEmployeeDAO(conn, handler);
```

6.2.4. Loggerクラス

Logger クラスには次の 12 のメソッドが存在します。

- fatal(int event, Object[] param)
- error(int event, Object[] param)
- warn(int event, Object[] param)
- info(int event, Object[] param)
- debug(int event, Object[] param)
- trace(int event, Object[] param)
- isFatalEnabled() : boolean
- isErrorEnabled() : boolean
- isWarnEnabled() : boolean
- isInfoEnabled() : boolean
- isDebugEnabled() : boolean
- isTraceEnabled() : boolean

このうち、自動生成される DAO のメソッド内部で使用されるのは、「trace」、「error」、「isTraceEnabled」、「isErrorEnabled」の 4 つです。「trace」と「isTraceEnabled」は正常処理時に、「error」と「isErrorEnabled」は異常処理時に呼び出されます。他のメソッドは将来の拡張用、および、「カスタム・メソッド」用として、宣言されています。引

数 event は、はメソッドが呼び出された時点の状態を表します。event には次のいずれかが渡されます。

状態	説明
BEFORE_EXECUTE	SQL 発行前の状態。
AFTER_EXECUTE	SQL 発行後、結果表へのアクセスも全て完了した状態。
ERROR	DB アクセス時に何らかの例外が発生した場合。

表 6-5 Logger が扱う状態一覧

また param に渡される配列の各要素の意味は次のとおりです。

要素番号	説明
0	呼び出し元(通常は DAO)のスレッド名。具体的には Thread # getName()メソッドの戻り値。
1	呼び出し元(通常は DAO)のクラス名。具体的には Class # getName()メソッドの戻り値。
2	呼び出し元(通常は DAO)のメソッド名。たとえば、「find」、「select」など。
3	実行前、あるいは実行後の SQL-ID。
4	SQL-ID に対応する SQL。
5	Parameter クラスのオブジェクト。Parameter を使用していないメソッドでは null。
6	最大照会件数。最大件数指定が行われていないメソッドでは null。
7	メソッドの実行結果。実行失敗時には例外オブジェクトか、null。照会処理が成功した場合は DTO(または Map)のインスタンス、または、DTO インスタンスのリスト。更新処理が成功した場合は更新件数。
8	SQL 実行時間(ms)。実際には JDBC/SQLJ の SQL 発行前から結果を全て取得(DTO へのマッピング)までの経過時間。

表 6-6 Logger に渡される情報一覧

DAO 生成ツールでは、Logger クラスのデフォルト実装として以下の 3 つの実装クラス(DefaultLogger クラス)を提供しています。

実装タイプ	設定方法
JDKLogger を使用した実装 (デフォルト)	コード生成時に、プロバイダー「com.ibm.jp.daogen.codegen.db.DefaultLoggerProvider」のプロパティー「impl_type」で「Default」と設定した場合に出力される。
ログ出力機能を使用した実装	V7.0.5 以降で選択可能。 コード生成時に、プロバイダー「com.ibm.jp.daogen.codegen.db.DefaultLoggerProvider」のプロパティー「impl_type」で「wacs」と設定した場合に出力される。 動作にはログ出力機能の前提となるライブラリーを用意する必要がある。
ログ出力機能(CommonsLogging)および Spring を使用した実装	V7.0.5 以降で選択可能。 コード生成時に、プロバイダー「com.ibm.jp.daogen.codegen.db.DefaultLoggerProvider」のプロパティー「impl_type」で「commons-logging」と設定した場合に出力される。 動作にはログ出力機能(CommonsLogging)の前提となるライブラリーを用意する必要がある。

表 6-7 Logger のデフォルト実装

一方、独自の Logger を DAO に設定することも可能です。*DAOName*DAOFactory # get*DAOName*DAO(Connection, Logger)メソッドを使用して DAO インスタンスを取得し、第二引数に独自の Logger クラスを指定してください。

```
// Logger クラスの宣言(匿名内部クラス)
Logger logger = new DefaultLogger() {
   public void trace(int event, Object[] param) {
      // 独自処理
   }
}

// Connection 取得(任意の方法)
Connection conn = getConnection(args);

// Factory 生成
EmployeeDAOFactory factory = EmployeeDAOFactory.getFactory();
```

```
// DAO インスタンス取得
EmployeeDAO dao = factory.getEmployeeDAO(conn, logger);
```

以下に、JDKLogger を使用した場合のログ出力例を示します。

```
2010/10/26 18:38:33 sample.impl.DefaultLogger trace
最高レベルのトレース情報:
ThreadID=main/ClassName=sample.impl.EmployeeCustomDAO/MethodName=find/SQLID=Employee_SEL
ECT/SQL=SELECT EMP_ID, EMP_NAME, LAST_DATE, LAST_TIME, LAST_OP_ID FROM EMPLOYEE
WHERE EMP_ID = ?/Parameter={EMP_ID=someID}/MaxCount=0
2010/10/26 18:38:33 sample.impl.DefaultLogger trace
最高レベルのトレース情報:
ThreadID=main/ClassName=sample.impl.EmployeeCustomDAO/MethodName=find/SQLID=Employee_SEL
ECT/SQL=SELECT EMP_ID, EMP_NAME, LAST_DATE, LAST_TIME, LAST_OP_ID FROM EMPLOYEE
WHERE EMP_ID = ?/Parameter={EMP_ID=someID}/MaxCount=0/Result={emp_id=someID    ,
emp_name=someName                    , last_date=2010-10-26, last_time=00:00:00,
last_op_id=someID    }/Elapsed=266
```

DefaultLogger クラスには、boolean を引数とするコンストラクターが用意されています。引数に true を指定して DefaultLogger のインスタンスを生成した場合、SQL パラメーター値のマスキング機能が有効になります。

以下に、マスキング機能を有効にした場合のログ出力例を示します。

```
2010/10/26 18:40:36 sample.impl.DefaultLogger trace
最高レベルのトレース情報:
ThreadID=main/ClassName=sample.impl.EmployeeCustomDAO/MethodName=find/SQLID=Employee_SEL
ECT/SQL=SELECT EMP_ID, EMP_NAME, LAST_DATE, LAST_TIME, LAST_OP_ID FROM EMPLOYEE
WHERE EMP_ID = ?/Parameter=*/MaxCount=0
2010/10/26 18:40:36 sample.impl.DefaultLogger trace
最高レベルのトレース情報:
ThreadID=main/ClassName=sample.impl.EmployeeCustomDAO/MethodName=find/SQLID=Employee_SEL
ECT/SQL=SELECT EMP_ID, EMP_NAME, LAST_DATE, LAST_TIME, LAST_OP_ID FROM EMPLOYEE
WHERE EMP_ID = ?/Parameter=*/MaxCount=0/Result={emp_id=someID    ,
emp_name=someName                    , last_date=2010-10-26, last_time=00:00:00,
last_op_id=someID    }/Elapsed=235
```

7. その他の情報

7.1. Springとの連携

この章ではオープンソースの DI/AOP フレームワークである Spring と DAO 生成ツールの連携方法について説明します。

7.1.1. Springとは

Spring とは、他のオープンソースとの統合環境機能を持つ、オープンソースの DI/AOP（※）フレームワークです。DAO 生成ツールが生成する Spring 用 DAO クラスと Spring ライブラリーを連携させることにより、DAO 生成ツールの DAO クラスが持つ機能に加えて、さらに次の機能を利用することができます。

1) データソース管理
2) トランザクション管理(宣言的トランザクション)
3) DAO 実装インスタンスの生成/切り替え
4) DAO インスタンスの他クラスへの注入(設定)

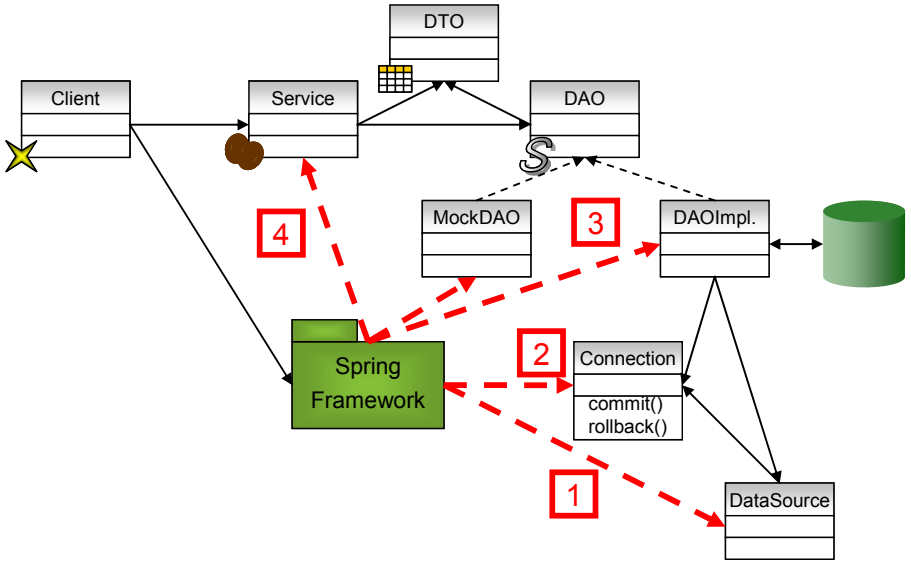


図 7-1 DAO 生成ツールと Spring

※ DI(Dependency Injection):コンポーネント同士の結びつき(Dependency)を下げるために、コンポーネントの生成や管理をプログラムから分離し、実行時に動的に結び付ける（Injection）仕組み。
※ AOP(Aspect Oriented Programming):アプリケーション内の複数のコンポーネントに存在する処理(ログ出力処理など)を分離し、実行時／コンパイル時に追加する仕組みです。

7.1.2. 事前準備

DAO 生成ツールが生成する DAO クラスと Spring を連携させるには、Spring ライブラリーをダウンロード(http://www.springframework.org/)、コンパイル環境および実行環境へ導入する必要があります。また、『プロバイダー設定ウィザード』上で「Spring(JDBC)」を指定し、Spring用のDAOクラスを生成する必要があります。詳細は5 コード生成をご参照ください。

まず、注入先クラスの宣言に@Service アノーテーション/@Transactional アノーテーションを記述してください。これにより、Spring が DAO クラスの注入先を判別できるようになり、また、宣言的トランザクションが利用可能となります。また、DAO クラスのプロパティーと setter メソッドを記述してください。さらに、setter メソッドに対して@Autowired アノーテーションを記述してください。これにより、setter メソッドに対して DAO クラスが設定されるようになります。

```
@Service
@Transactional
public class OriginalService {
        …
```

```
        private EmployeeDAO empDao = null;
        ・・・
        @Autowired
        public void setEmployeeDao(EmployeeDAO empDao) {
                this.empDao = empDao;
        }
        ・・・
}
```

次に、Spring 設定ファイルを作成します。Spring 設定ファイルには注入先クラス/DAO クラスのパッケージ情報、データソース情報、トランザクション設定を記述します。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xsi:schemaLocation="
                http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                http://www.springframework.org/schema/tx
                http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context-2.5.xsd
        ">

<!-- パッケージ情報 -->
        <context:component-scan base-package="注入先クラスのパッケージ名,DAO クラスのパッケージ名"
/>

<!-- DAO 用ログ情報 -->
        <bean id=”logger” class=”DefaultLogger クラス名” />

<!-- データソース情報 (例は DB2 かつ DriverManager 経由でコネクションを取得する場合) -->
        <bean id="dataSource"
                class="org.apache.commons.dbcp.BasicDataSource"
                destroy-method="close">
                <property name="driverClassName">
                        <value>com.ibm.db2.jcc.DB2Driver</value>
                </property>
                <property name="url">
                        <value>jdbc:db2://localhost:50000/sample</value>
                </property>
                <property name="username">
                        <value>ユーザー名</value>
                </property>
                <property name="password">
                        <value>パスワード</value>
                </property>
        </bean>

<!-- トランザクション設定 -->
        <tx:annotation-driven transaction-manager="txManager" />
        <bean id="txManager"
                class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
                <property name="dataSource" ref="dataSource" />
        </bean>
</beans>
```

なお、各アノーテーションの詳細や Spring 設定ファイル上での注入方法については、Spring のリファレンス・ガイドをご参照ください。

7.1.3. 利用方法

注入先クラスの呼び出し方法は、Spring の仕様に従い、BeanFactory クラス経由で取得する必要があります。

```
BeanFactory factory = new ClassPathXmlApplicationContext(Spring 設定ファイルのクラスパス);
OriginalService bean = (OriginalService) factory.getBean("originalService");
```

このBeanFactory経由で注入先クラスのインスタンスを取得することにより、Springライブラリーが7.1.2　事前準備にて指定した@Serviceアノーテーションを認識し、注入先クラスにDAOインスタンスが注入されます。

また、7.1.2　事前準備にて指定した@Transactionalアノーテーションにより、注入先クラスのメソッド内でjava.lang.RuntimeExceptionおよびそのサブクラスがthrowされた場合、トランザクションが自動的にロールバックされます。これを変更するには、変更対象のメソッドに@Transactionalアノーテーションをさらに追加します。

```
@Service
@Transactional
public class OriginalService {
	・・・
	private EmployeeDAO empDao = null;
	・・・
	@Autowired
	public void setEmployeeDao(EmployeeDAO empDao) {
		this.empDao = empDao;
	}
	・・・
	@Transactional(rollbackFor={DataAccessException.class})
	public Object execute(Object arg) throws DataAccessException {
		// DAO クラスを使用した DB アクセス処理
	}
	・・・
}
```

@Transactional アノーテーションの詳細は、Spring のリファレンス・ガイドをご参照ください。

なお、単体テスト等でモック DAO クラスを使用する場合、次のような Spring 設定ファイルを別途作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:tx="http://www.springframework.org/schema/tx"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
		http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
		http://www.springframework.org/schema/tx
		http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
		http://www.springframework.org/schema/context
		http://www.springframework.org/schema/context/spring-context-2.5.xsd
	">
	<bean class="注入先クラス名" scope="prototype">
		<property name="employeeDao">
			<bean class="モック DAO クラス名"/>
		</property>
	</bean>
</beans>
```

単体テスト時 BeanFactory 取得時にこの Spring 設定ファイルを使用することにより、モック DAO クラスが注入先 DAO クラスに注入されます。これは、注入先クラスの setter メソッドに指定された@Autowired アノーテーションより、Spring 設定ファイル上での設定が優先されるためです。

```
BeanFactory factory = new ClassPathXmlApplicationContext(Spring 設定ファイルのクラスパス);
OriginalService bean = (OriginalService) factory.getBean("originalService");
```

7.2. Excel設計書との連携

この章では本アセットで提供する DTO 設計書テンプレートや DAO 設計書テンプレートに DTO 仕様や DAO 仕様を記述し、DAO 生成ツール上で直接設計書をインポートして DAO を生成する方法について説明します。

7.2.1. Excel設計書との連携に使用するExcelファイルの取得方法

本アセットで提供する DAO 生成ツールの Excel 設計書との連携機能で使用する Excel ファイルは、以下のディレクトリーから取得してください。

Base Edition の場合
%RAD のプラグイン・インストール・ディレクトリー%¥com.ibm.jp.wacs.contents.common_(バージョン番号)¥DAOGenerator¥excel

Express Edition の場合
%RAD のプラグイン・インストール・ディレクトリー%¥com.ibm.jp.wacs.contents.common.exp_(バージョン番号)¥DAOGenerator¥excel

7.2.2. Excel設計書によるDAO生成の流れ

Excel 設計書との連携機能は、DDL からの DTO 設計書の自動生成機能と、DTO 設計書/DAO 設計書を DAO 生成ツールにインポートする機能を含みます。

テーブル情報を定義する DDL と DTO 設計書テンプレートを本アセットで提供する DTO 設計書生成ツール(Excel マクロ)に読み込ませることで、DTO 設計書を DDL に含まれるテーブル定義の数だけ自動生成することができます。
自動生成された DTO 設計書では、DDL で定義されているカラム情報が反映されます。

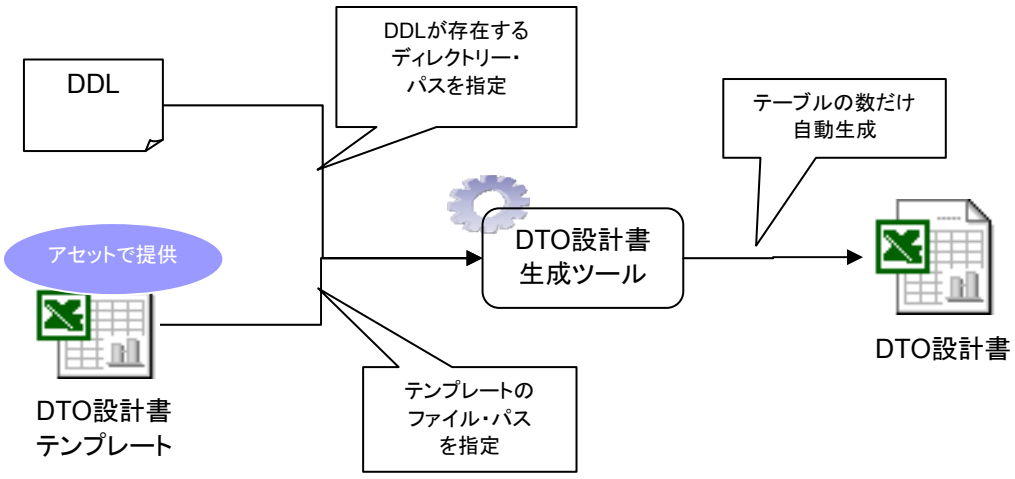


図 7-2 DTO 仕様書の自動生成の流れ

DTO 設計書を自動生成した後、必要であれば DTO 設計書に対し DTO 関連項目の記述を行います。
DTO 設計書が完成した後、開発者は DAO 設計書テンプレートをコピーして DAO 仕様書を編集します。
DTO 設計書と DAO 設計書を DAO 生成ツールに直接インポートすると、DAO 生成ツール上の定義に変換されますので、その後は DTO クラス/DAO クラスをそのまま自動生成できます。

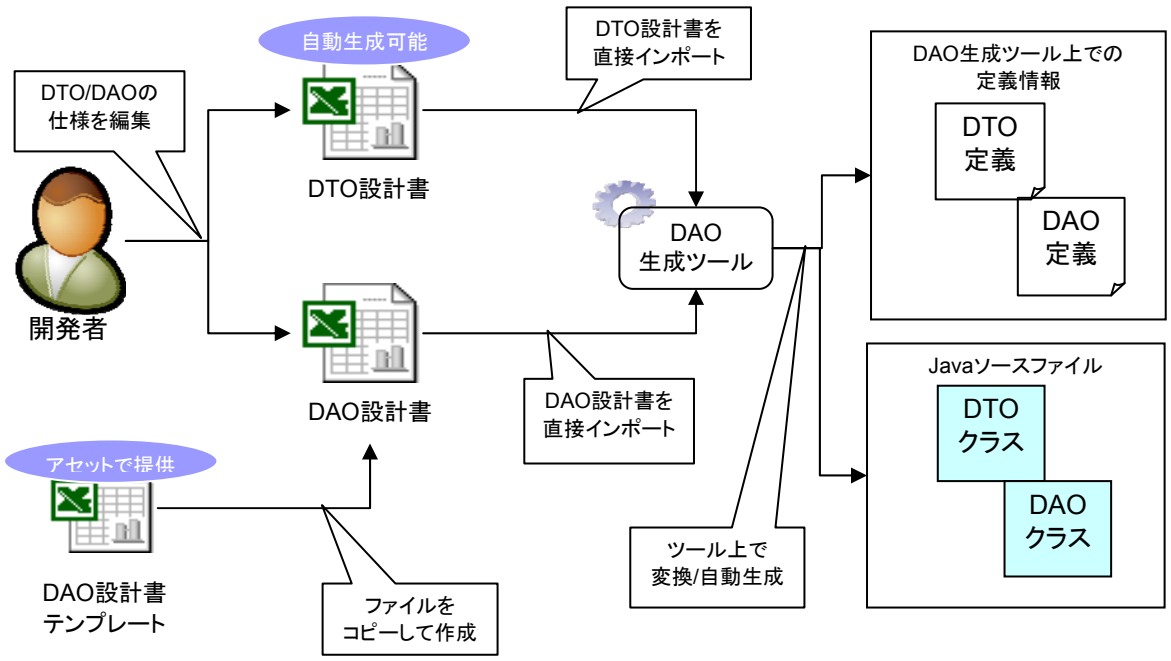


図 7-3 DTO クラス/DAO クラスの自動生成の流れ

7.2.3.DTO設計書テンプレート

DTO 設計書テンプレートのシート構成は次の通りです。シート名の変更はできません。

シート名	説明
表紙	DTO の基本的な定義情報を記述します。
DTO 定義	DTO の各フィールドに対する詳細な定義を記述します。
DTO 関連項目	フィールドに対し関連 DTO が存在する場合にその定義を記述します。

表 7-1 DTO 設計書テンプレートの構成

DTO 設計書テンプレートの表紙シートの中で DAO 生成ツールが参照する項目は次の通りです。

項目名	説明
DB 種別	DB2 または Oracle を選択します。
パッケージ名	ソースコードの自動生成先のパッケージ名を記述します。
DTO 名	DTO のクラス名を記述します。
表名	DTO の表名を記述します。
スキーマ名	表のスキーマ名を記述します。

表 7-2 DTO 設計書の表紙シートの項目

DTO 設計書テンプレートの表紙シートの記述例を以下に示します。

表紙:DTO設計書

システム名	サブシステム名	作成者	作成日	更新者	更新日	承認者	承認日
xxxシステム	xxx	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd

DB種別	DB2
パッケージ名	com.ibm.jp.daogen
DTO名	Department
表名	DEPARTMENT
スキーマ名	db2admin

変更履歴	

図 7-4 DTO 設計書テンプレートの表紙シートの記述例

DTO 設計書テンプレートの DTO 定義シートの中で DAO 生成ツールが参照する項目は次の通りです。

項目名	説明
列名	テーブルの列名を記述します。
列データ型	列のデータ型を DB2 または Oracle でサポートしているデータ型の中から選択します。
主キー	主キーであるかどうかを YES または NO で選択します。
NOT NULL	NOT NULL であるかどうかを YES または NO で選択します。
自動生成列	自動生成列であるかどうかを YES または NO で選択します。
フィールド名	DTO のフィールド名を記述します。
Java データ型	DTO フィールドの Java データ型を選択します。
初期値	DTO フィールドの初期値を記述します。

表 7-3 DTO 設計書テンプレートの DTO 定義シートの項目

DTO 設計書テンプレートの DTO 定義シートの記述例を以下に示します。

システム名	サブシステム名	DTO仕様	作成者	作成日	更新者	更新日	承認者	承認日
xxxシステム	xxx	Department	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd

コメント	2011年x月x日：xxxWebで追加

予備1	
予備2	
予備3	
予備4	

列名	列データ型	主キー	NOT NULL	自動生成列	フィールド名	Javaデータ型	初期値	予備列1	予備列2
DEPTNO	CHARACTER	YES	YES	NO	deptno	java.lang.String	""		
DEPTNAME	VARCHAR	NO	NO	NO	deptname	java.lang.String	""		
MGRNO	CHARACTER	NO	YES	NO	mgrno	java.lang.String	""		
ADMRDEPT	CHARACTER	NO	NO	NO	admrdept	java.lang.String	""		
LOCATION	CHARACTER	NO	YES	NO	location	java.lang.String	""		

図 7-5 DTO 設計書テンプレートの DTO 定義シートの記述例

DTO 設計書テンプレートの DTO 関連項目シートの中で DAO 生成ツールが参照する項目は次の通りです。

項目名	説明
フィールド名	関連を記述する対象となる DTO フィールド名を記述します。
関連	関連を 1:1 または 1:n から選択します。
関連 DTO 名	関連 DTO のクラス名をパッケージ名を含めて記述します。

表 7-4 DTO 設計書テンプレートの DTO 関連項目シートの項目

DTO 設計書テンプレートの DTO 関連項目シートの記述例を以下に示します。

システム名	サブシステム名	DTO仕様	作成者	作成日	更新者	更新日	承認者	承認日
xxxシステム	xxx	Department	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd

フィールド名	関連	関連DTO名
manager	1:1	com.ibm.jp.daogen.Employee
employee	1:n	com.ibm.jp.daogen.Employee
assistant	1:1	com.ibm.jp.daogen.Employee

図 7-6 DTO 設計書テンプレートの DTO 関連項目シートの記述例

7.2.4. DAO設計書テンプレート

DAO 設計書テンプレートのシート構成は次の通りです。

シート名	説明
表紙	DAO に対する基本的な定義情報を記述します。表紙のシート名は変更しないで下さい。

メソッド仕様	DAO の各メソッドの定義情報を記述します。1 つの DAO に対し複数のメソッドを定義する場合は、1 つのワーク・ブックに対してこのシートを複数コピーして追加することで記述します。シート名は「メソッド仕様」という接頭語を付加してください。

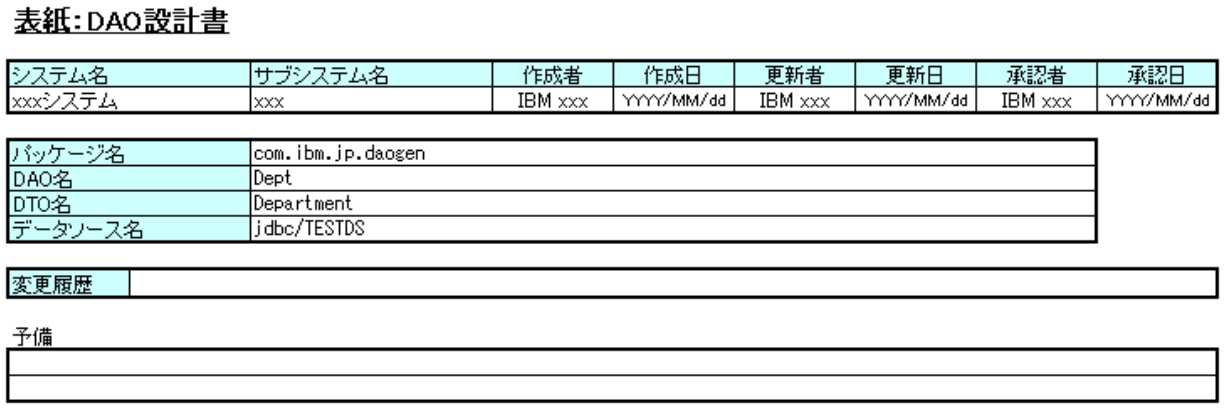
表 7-5 DAO 設計書テンプレートの構成

DAO 設計書テンプレートの表紙シートの中で DAO 生成ツールが参照する項目は次の通りです。

項目名	説明
パッケージ名	ソースコードの自動生成先のパッケージ名を記述します。
DAO 名	DAO のクラス名を記述します。自動生成時にここで指定したクラス名に対して「DAO」という文字列がインターフェース名の末尾に付加されます。 例：記述した DAO クラス名が「Dept」の場合、自動生成されるインターフェース名は「DeptDAO」となる
DTO 名	DAO が使用する DTO のクラス名を記述します。
データソース名	DAO が使用するデータソースの JNDI 名を記述します。

表 7-6 DAO 設計書の表紙シートの項目

DAO 設計書テンプレートの表紙シートの記述例を以下に示します。



表紙：DAO設計書

システム名	サブシステム名	作成者	作成日	更新者	更新日	承認者	承認日
xxxシステム	xxx	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd

パッケージ名	com.ibm.jp.daogen
DAO名	Dept
DTO名	Department
データソース名	jdbc/TESTDS

変更履歴	

予備

図 7-7 DAO 設計書テンプレートの表紙シートの記述例

DAO 設計書テンプレートのメソッド仕様シートの中で DAO 生成ツールが参照する項目は次の通りです。

項目名	説明
SQLID	メソッドの SQLID を記述します。
メソッド名	メソッド名を記述します。ここに記述したメソッド名がそのまま DAO クラスに生成されるメソッド名になります。ただし、カーソル更新およびカーソル削除のメソッド名は反映されず、update メソッドと delete メソッドでコード生成されます。 なお、V7.0.6 ではここに記述したメソッド名はコード生成時に利用されず、DAO生成ツールは表 4-5 照会系メソッド一覧, 表 4-6 更新系メソッド一覧にある命名規則に基づいたメソッド名でコード生成する仕様でしたが、V7.1.1 以降ではDAO設計書に記述したメソッド名がそのままソースコードに反映されます。ただし、メソッド名の記述がない場合は、従来どおりDAO生成ツールの命名規則に基づいたメソッド名でコード生成されます。
ステートメント種別	ステートメント種別を PreparedStatement または Statement から選択します。
メソッド詳細	DAO 生成ツールがサポートするメソッドの種別を選択リストの中から選択します。
UPDATE	メソッド詳細が「カーソル照会」の場合に、カーソル更新を行うメソッドの SQLID を記述します。カーソル更新のメソッド仕様は同一ワーク・ブック内に

	記述してください。
DELETE	メソッド詳細が「カーソル照会」の場合に、カーソル削除を行うメソッドの SQLID を記述します。カーソル削除のメソッド仕様は同一ワーク・ブック内に記述してください。
SQL 文	実行する SQL 文を記述します。テンプレートが用意する行数で足りない場合は、既存の行を行ごとコピーして行数を増やしてください。
パラメーター（パラメーター項目一覧）	SQL 文の中で使用しているパラメーター名を記述します。
Java データ型/DTO フィールド名(パラメータ項目一覧)	メソッドの引数が Parameter 型の場合には、Java のデータ型を選択します。メソッドの引数が DTO の場合には、DTO のフィールド名を記述します。関連 DTO のフィールドを指定する場合は、関連 DTO を表すフィールド名と DTO のフィールド名を「-」(ハイフン)で区切って記述します。 例：「CompnayDTO-EmployeeDTO-DepartmentDTO-deptname」の場合「employee-department-deptname」と記述
取得列（取得列項目一覧）	SQL 文の中で使用している取得列を記述します。
Java データ型/DTO フィールド名(取得列項目一覧)	メソッドの戻り値が Map 型の場合には、Java のデータ型を選択します。メソッドの引数が DTO の場合には、DTO のフィールド名を記述します。関連 DTO のフィールドを指定する場合は、関連 DTO を表すフィールド名と DTO のフィールド名を「-」(ハイフン)で区切って記述します。 例：「CompnayDTO-EmployeeDTO-DepartmentDTO-deptname」の場合「employee-department-deptname」と記述

表 7-7 DAO 設計書のメソッド仕様シートの項目

DAO 設計書テンプレートのメソッド仕様シートの記述例を以下に示します。

システム名	サブシステム名	DAOメソッド仕様	作成者	作成日	更新者	更新日	承認者	承認日
xxxシステム	xxx	deptSelect01	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd	IBM xxx	YYYY/MM/dd

SQLID	Department_Select01			
メソッド名				
ステートメント種別	PreparedStatement			
メソッド詳細	単一行照会	UPDATE	DELETE	

コメント	

予備1	
予備2	
予備3	
予備4	

SQL文	SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION, EMPNO FROM DEPARTMENT, EMPLOYEE WHERE DEPTNO = WORKDEPT AND DEPTNO = :DEPTNO AND EMPNO = 'xxx'

パラメータ項目一覧

パラメーター	Javaデータ型/DTOフィールド名	予備1	予備2
DEPTNO	java.lang.String		

取得列項目一覧

取得列	Javaデータ型/DTOフィールド名	予備1	予備2
DEPTNO	deptno		
DEPTNAME	deptname		
MGRNO	mgrno		
ADMRDEPT	admrdept		
LOCATION	location		
EMPNO	employee-empno		

図 7-8 DAO 設計書テンプレートのメソッド仕様シートの記述例

7.2.5.DTO設計書生成ツール

DTO 設計書生成ツールは、テーブルを作成するための CREATE TABLE 文が記述された DDL ファイルと DTO 設計書テンプレート・ファイルから、テーブルごとに必要な項目が記述された DTO 設計書を生成します。DDL ファイルはファイル拡張子を*.sql としてください。DDL 保存ディレクトリーには複数の*.sql ファイルを配置して DTO 設計書を一括生成することが出来ます。

DTO 設計書生成ツールの生成シートのユーザー・インターフェースを下記に示します。

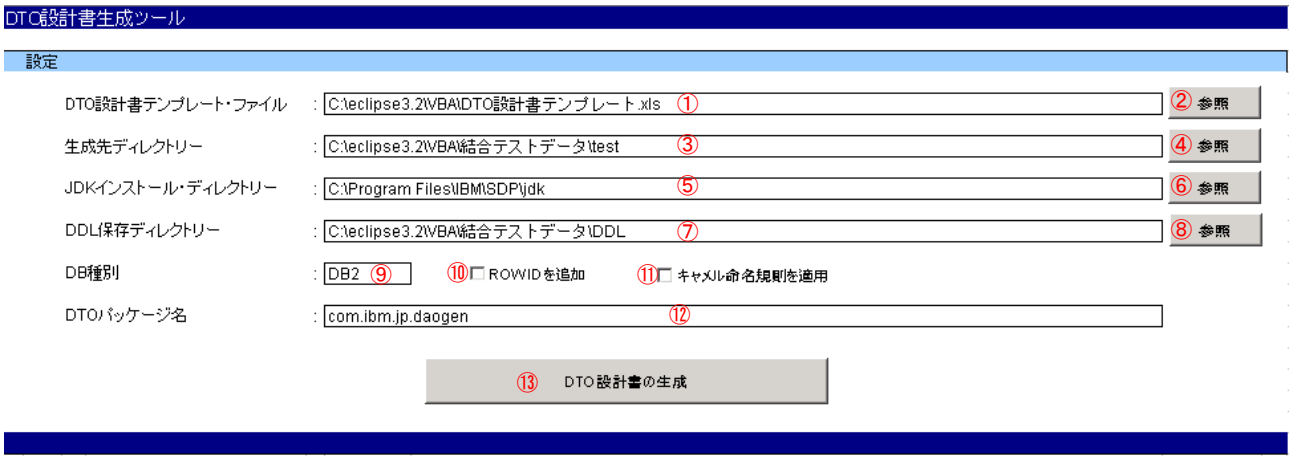


図 7-9 DTO 設計書生成ツールの生成シート

DTO 設計書生成ツールのユーザー・インターフェースの各部品の説明は次の通りです。

番号	項目名	説明
①	DTO 設計書テンプレート・ファイル	DTO 設計書テンプレート・ファイルのファイル・パスを記述します。ここで指定されたテンプレートに対して DDL から読み込んだ情報が転記されます。
②	参照ボタン	このボタンにを押下するとファイル選択ダイアログが表示されます。DTO 設計書テンプレート・ファイルのファイル・パスを指定する際に使用します。
③	生成先ディレクトリー	DTO 設計書を出力するディレクトリーのパスを記述します。
④	参照ボタン	このボタンを押下するとフォルダー参照ダイアログが表示されます。生成先ディレクトリーを指定する際に使用します。
⑤	JDK インストール・ディレクトリー	本ツールでは Java バッチ・プログラムを実行するため、Java を実行するための JDK インストール・ディレクトリーを記述します。
⑥	参照ボタン	このボタンを押下するとフォルダー参照ダイアログが表示されます。JDK インストール・ディレクトリーを指定する際に使用します。
⑦	DDL 保存ディレクトリー	DTO 設計書を生成したいテーブルを作成するための DDL が保存されているディレクトリーを記述します。
⑧	参照ボタン	このボタンを押下するとフォルダー参照ダイアログが表示されます。DDL 保存ディレクトリーを指定する際に使用します。
⑨	DB 種別	DB 種別を DB2 または Oracle で選択します。この項目は生成される全ての DTO 設計書に反映されます。
⑩	ROWID を追加	DTO に対し ROWID を追加する場合はチェックボックスにチェックを入れます。生成される全ての DTO 設計書に対し ROWID が追加されます。
⑪	キャメル命名規則を適用	DTO に対しキャメル命名規則を適用する場合はチェックボックスにチェックを入れます。この項目は生成される全ての DTO 設計書に反映されます。
⑫	DTO パッケージ名	ソースコードの自動生成先のパッケージ名を記述します。この項目は生成される全ての DTO 設計書に反映されます。
⑬	DTO 設計書の生成ボタン	このボタンを押下すると DTO 設計書が生成されます。①、③、⑤、⑦の項目が全て正しく指定されていることを確認後にこのボタンを押下してください。

表 7-8 DTO 設計書生成ツールの生成シートの項目

DTO 設計書の生成ボタンを押下すると、以下のような進捗ダイアログが表示され、DTO 設計書の生成が開始します。

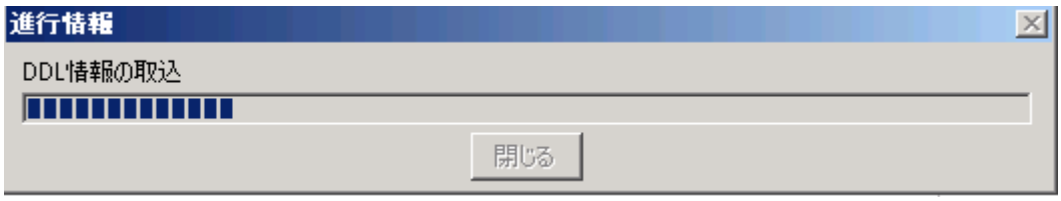


図 7-10 DTO 設計書生成ツールの進行情報ダイアログ

DTO 設計書の生成が正常に完了した場合は以下の表示になりますので、閉じるボタンを押下して終了してください。生成が正常に完了しない場合は、DTO 設計書生成ツールへの設定情報または DDL の記述方法に不備がある可能性があります。その場合は設定の確認や DDL の記述内容の確認を行ってください。

図 7-11 DTO 設計書生成ツールの設計書生成完了時の表示

7.2.6. DAO生成ツール上でのExcelファイルのインポート

作成した DTO 設計書/DAO 設計書を DAO 生成ツール上でインポートするには、DAO 生成ツール起動画面のアウトライン・ビューで DAO のパッケージを右クリックしてメニューを表示し、「Excel をインポート」の項目を選択してください。

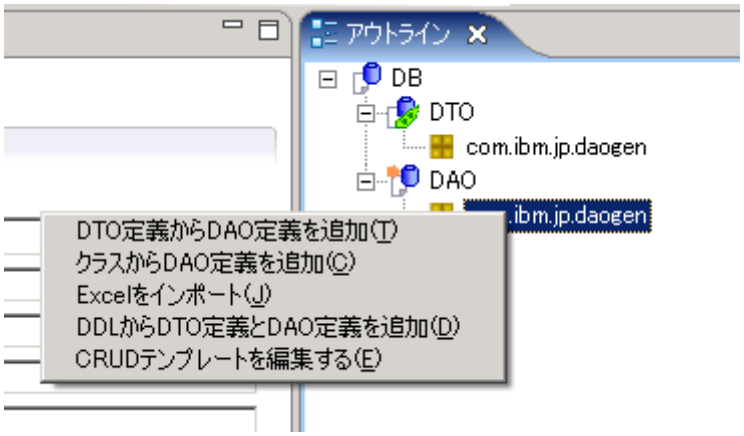


図 7-12 DAO 生成ツールの Excel インポート

「Excel をインポート」を選択すると、以下の Excel インポート・ダイアログが表示されます。

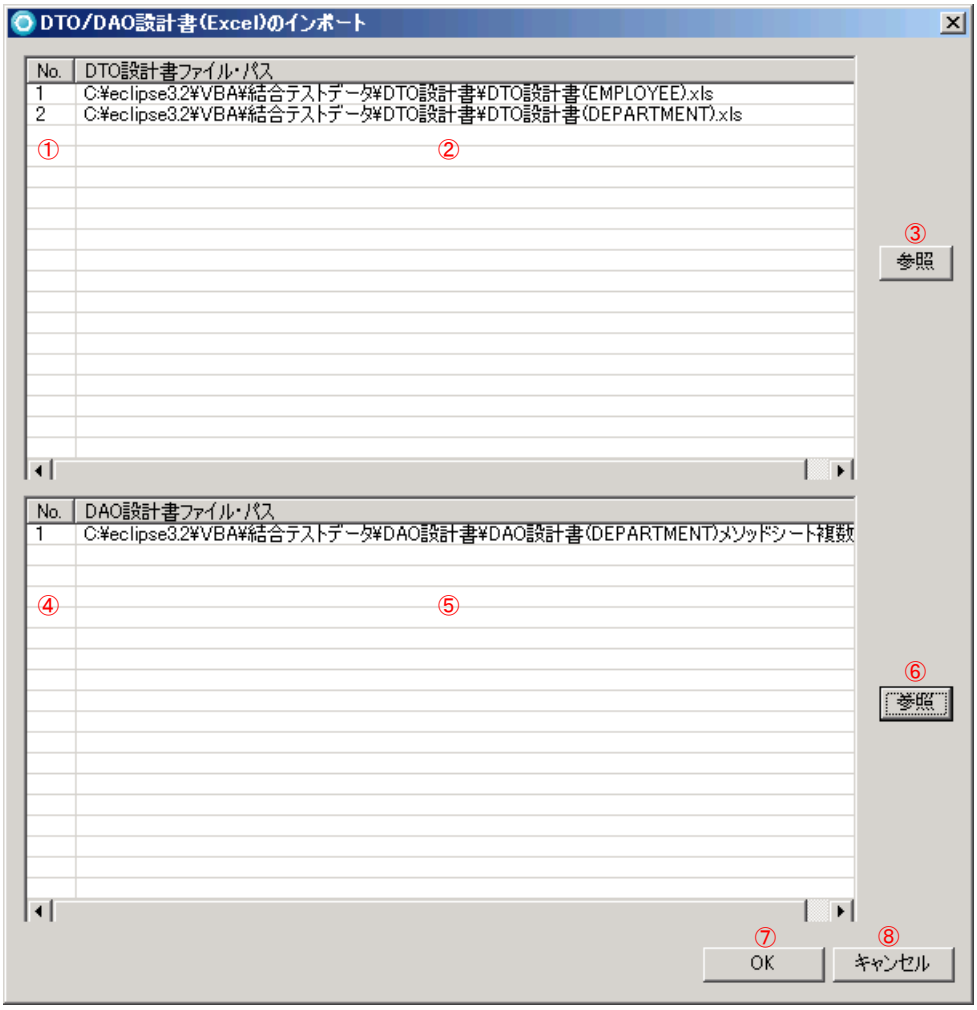


図 7-13 DAO 生成ツールの Excel インポート・ダイアログ

DAO 生成ツールの Excel インポート・ダイアログの各部品の説明は次の通りです。

番号	項目名	説明
①	No.	DTO 設計書の番号を表示します。
②	DTO 設計書ファイル・パス	DTO 設計書が存在するファイル・パスを表示します。
③	参照ボタン	このボタンを押下すると、ファイルの参照ダイアログが表示されます。同一フォルダー内にある DTO 設計書を Shift キーや Ctrl キーを押しながら複数指定することで一括生成することが可能です。
④	No.	DAO 設計書の番号を表示します。
⑤	DAO 設計書ファイル・パス	DAO 設計書が存在するファイル・パスを表示します。
⑥	参照ボタン	このボタンを押下すると、ファイルの参照ダイアログが表示されます。同一フォルダー内にある DAO 設計書を Shift キーや Ctrl キーを押しながら複数指定することで一括生成することが可能です。
⑦	OK ボタン	このボタンを押下すると、Excel の設計書から DAO 生成ツール上の DTO 定義/DAO 定義への変換が実行されます。DTO 設計書または DAO 設計書が 1 つも指定されていない場合は非活性化された状態となり、インポートを実行することはできません。
⑧	キャンセルボタン	このボタンを押下すると Excel インポート画面が閉じます。

表 7-9 Excel インポート・ダイアログの各部品の説明

Excel インポート・ダイアログの OK ボタンを押下すると、下記の進行情報ダイアログが表示され、Excel から DAO 生

成ツール上での DTO/DAO 定義への変換が始まります。このときにキャンセル・ボタンを押下すると、変換の途中で処理を中断することが可能です。処理を中断した場合は、DTO/DAO 定義が最後まで正しく作成されていない可能性がありますので、再度 Excel のインポートを実行してください。

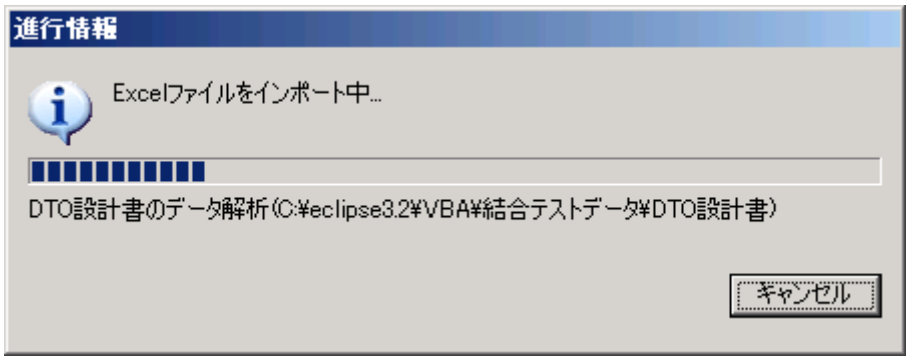


図 7-14 Excel インポート実行時の進行情報ダイアログ

Excel のインポートが成功すると、Excel インポート・ダイアログが閉じ、下記のように指定した DTO/DAO 設計書の数だけ DTO/DAO 定義が追加されます。Excel インポート・ダイアログが閉じないか、または DTO/DAO 定義が指定した DTO/DAO 設計書の数と合っていない場合、DTO/DAO 設計書のフォーマットまたは記述内容に不備がある可能性があります。その場合は DTO/DAO 仕様書のフォーマットと記述内容の確認を行ってください。

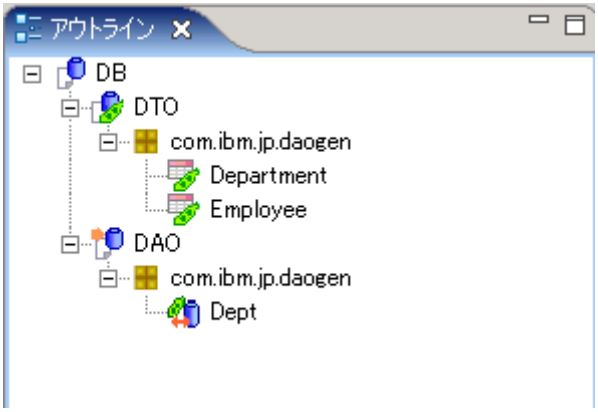


図 7-15 Excel インポート実行後のアウトライン・ビュー

7.2.7.DTO/DAO設計書テンプレートのカスタマイズ

DTO/DAO 設計書テンプレートは、本アセットが提供する Excel マクロが参照するデータ行を、各シートの入力欄の右の列（デフォルトで灰色表示のセル）に明記しています。この列をマクロ使用列と呼びます。マクロ使用列に記述があるセルの行は、DAO 生成ツールにより処理される領域です。また、マクロ使用列に記述されている文字列はデータ種別を表しています。各シートのマクロ使用列にデータ種別が記述されていない行（ヘッダー部分やフッター部分）については、自由にフォーマットの変更が可能です。

マクロ使用列は Excel の機能により非表示列にすることが可能です。また、デフォルトの列位置から変更することが可能です。その場合は列ごと切り取り、移動したい列の位置に挿入してください。ただし、マクロ使用列に記述されている「マクロ使用列」というヘッダーや各データ種別が出現する順番については変更ができませんので、マクロ参照列を直接変更しないようにしてください。

Excel マクロが参照する行であっても、複数行にわたるデータ種別の場合は、以下のいずれかの方法で記述可能な行数を増やすことができます。

- 同一データ種別の行の途中に行を挿入する
- 最終行の下に同一データ種別の行をコピー＆ペーストする

下記の図の例では、データ種別「sql」のデータ領域に複数行を挿入して記述行数を増やしています。本アセットが提供する Excel マクロは、マクロ使用列にデータ種別（この例では「sql」）が記述されている最終行まで定義情報として認識し、参照を行います。

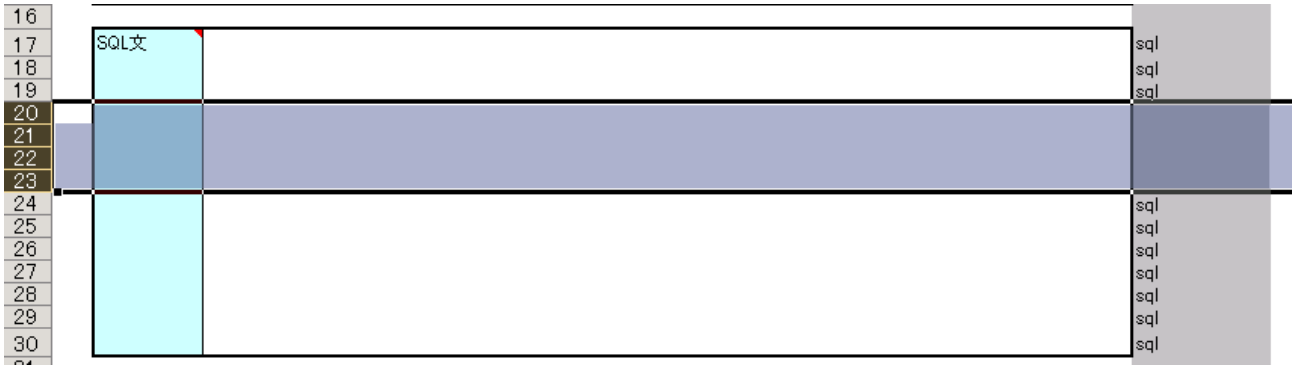


図 7-16 データ種別「sql」の途中行に行を挿入して記述可能行数を増やした場合

7.2.8. Excelインポートにおける注意事項

サンプルに含まれる DTO 設計書および DAO 設計書はパッケージが「com.ibm.jp.daogen」で記述されています。インポートに失敗する場合は、パッケージ名の整合性の確認をしてください。
DTO 設計書や DAO 設計書ファイル名およびパスに半角スペースを使用できません。また、ファイル名とパスの合計を 218 バイト以内としてください。これらの条件のいずれかに合致する場合、DAO 生成ツール上でのインポートに失敗します。
SQL 文として1つのセルに記述できる文字数は 255 文字までです。それ以上の長さの文字列を記述した場合、インポート時に各セルが 255 文字までの長さで切り捨てられてしまうため、SQL を正常に取り込めない可能性があります。

7.3. Ant

DAO 生成ツールの種々の機能を、Ant のタスクとして利用することができます。

7.3.1. DDL読み込み

Ant プロジェクト(build.xml)の記述例は次の通りです。

```
<project name="daogen" default="generate">
  <!-- タスク定義 -->
  <taskdef name="importDdl"
           classname="com.ibm.jp.daogen.db.ant. ImportDdlTask"
           classpath="WACs-Commons.jar;base.jar;daogendb.jar" />

  <!-- DDL 読み込み -->
  <target name="importDdl">
    <importDdl projectFile="./tool/sample.dgdx" providerGroup="Spring(JDBC)"
packageName=”com.ibm.jp.daogen.sample” dbms=”DB2” ddlPath=”./ddl” appendRowId=”false”
applyCamel=”false” importOnly=”true” />
  </target>
</project>
```

classpath 属性に指定する「WACs-Commons.jar」、「base.jar」と「daogendb.jar」は、それぞれ次のディレクトリーに配置されています。環境に合わせてパスを変更してください。

WACs-Commons.jar: *%Eclipse プラグイン・ディレクトリー%*￥com.ibm.jp.daogen_X.X.X/lib
base.jar: *%Eclipse プラグイン・ディレクトリー%*￥com.ibm.jp.daogen_X.X.X
daogendb.jar: *%Eclipse プラグイン・ディレクトリー%*￥com.ibm.jp.daogen.db_X.X.X

DDL 読み込みタスク(図中<importDdl>タグ)に指定する属性の説明は次の通りです。

属性	説明	必須
projectFile	DAO 生成設定ファイル(*.dgdx)のパスを指定します。	必須
providerGroup	生成する DAO 実装を選択します。指定できる値は次のいずれかです。 JDBC 実装:「JDBC」 SQLJ 実装:「SQLJ」	必須

	Spring(JDBC)による実装:「Spring(JDBC)」 Spring(SQLJ)による実装:「Spring(SQLJ)」	
dbms	DBMS 種別(DB2/Oracle)を指定します。	必須
packageName	読み込み結果となる DTO 定義の生成先パッケージ名を指定します。	必須
ddlPath	DDL ファイル(*.sql)が存在するディレクトリーを指定します。	必須
appendRowId	DDL ファイル読み込み後、DTO 定義に ROWID 列を追加するかどうかを指定します。デフォルト値は false(追加しない)です。	必須ではない
applyCamel	DDL ファイル読み込み後、DTO 定義の DTO クラスおよびフィールド名をキャメル記法に沿って変換するかどうかを指定します。デフォルト値は false(変換しない)です。	必須ではない
importOnly	DDL ファイル読み込み後、DTO 定義のみを作成するか、それを用いてコード生成を行うかどうかを指定します。デフォルト値は false(コード生成を行う)です。	必須ではない

表 7-10 DDL 読み込みタスクの属性

7.3.2. SQL読み込み

Ant プロジェクト(build.xml)の記述例は次の通りです。

```
<project name="daogen" default="generate">
  <!-- タスク定義 -->
  <taskdef name="importSql"
           classname="com.ibm.jp.daogen.db.ant. ImportSqlTask"
           classpath="WACs-Commons.jar;base.jar;daogendb.jar" />

  <!-- SQL 読み込み -->
  <target name="importSql">
    <importSql projectFile="./tool/sample.dgdx" providerGroup="Spring(JDBC)" sqlPath="./sql"
importOnly="true" />
  </target>
</project>
```

SQL 読み込みタスク(図中<importSql>タグ)に指定する属性の説明は次の通りです。

属性	説明	必須
projectFile	DAO 生成設定ファイル(*.dgdx)のパスを指定します。	必須
providerGroup	生成する DAO 実装を選択します。指定できる値は次のいずれかです。 JDBC 実装:「JDBC」 SQLJ 実装:「SQLJ」 Spring(JDBC)による実装:「Spring(JDBC)」 Spring(SQLJ)による実装:「Spring(SQLJ)」	必須
sqlPath	SQL ファイルが存在するディレクトリーを指定します。SQL ファイルは、指定したディレクトリーの配下に、次のルールに従って格納されている必要があります。 <パッケージ名>/<DAO 名>/<SQL-ID>(_<Statement Type>_<Method Type>_<Method Name>).sql Statement Type : ステートメント種別(ST/PS) Method Type : メソッド種別(「4.2 DAO定義情報の編集と操作」) Method Name : メソッド名 例) com.ibm.sample/Dept/SELECT_PS_SF_find1.sql	必須(*)

表 7-11 SQL 読み込みタスクの属性

7.3.3. コード生成

Ant プロジェクト(build.xml)の記述例は次の通りです。

```
<project name="daogen" default="generate">
  <!-- タスク定義 -->
  <taskdef name="generate"
           classname="com.ibm.jp.daogen.db.ant.GenerateDaoAndDtoTask"
           classpath="WACs-Commons.jar;base.jar;daogendb.jar" />

  <!-- コード生成 -->
  <target name="generate">
    <generate projectFile="./tool/sample.dgdx" providerGroup="Spring(JDBC)" classPath="./bin" />
  </target>
</project>
```

コード生成タスク(図中<generate>タグ)に指定する属性の説明は次の通りです。

属性	説明	必須
projectFile	DAO 生成設定ファイル(*.dgdx)のパスを指定します。	必須
providerGroup	生成する DAO 実装を選択します。指定できる値は次のいずれかです。 JDBC 実装:「JDBC」 SQLJ 実装:「SQLJ」 Spring(JDBC)による実装:「Spring(JDBC)」 Spring(SQLJ)による実装:「Spring(SQLJ)」	必須
classPath	既存の DTO クラスから DAO クラスを生成する場合、DTO クラスへのクラスパスを指定する必要があります。	必須ではない

表 7-12 コード生成タスクの属性

7.3.4. 実行可能SQLの生成

SQL 定義ファイル(sql.properties)には、SQL-ID や DAO 生成ツール独自の拡張 SQL(:_WHERE_AND など)が含まれているため、DBMS 上で直接実行することができません。この Ant タスクを使用することにより、SQL 定義ファイルを実行可能な SQL ファイルへ変換することができます。これを利用して DBMS の Explain 機能を利用することができます。Ant プロジェクト(build.xml)の記述例は次の通りです。

```
<project name="ExplainCheck" default="createSql">
  <!-- タスク定義 -->
  <taskdef name="createSql"
    classname="com.ibm.jp.daogen.db.ant.SqlPropertiesToFileTask"
    classpath="WACs-Commons.jar;base.jar;daogendb.jar" />

  <!-- 実行可能 SQL 作成 -->
  <target name="createSql">
    <createSql />
  </target>
</project>
```

<taskdef> タグの classname 属性に実行可能 SQL 作成用のクラス「com.ibm.jp.daogen.db.ant.SqlPropertiesToFileTask」を指定します。実行可能 SQL 作成タスク(図中<createSql>タグ)に指定する属性の説明は次の通りです。

属性	説明	必須
propFileName	SQL 定義ファイル名です。デフォルト値は「sql.properties」です。	必須ではない
destination	実行可能な SQL の出力先です。既存のフォルダーを指定した場合は、SQL-ID 毎に異なるファイルが出力されます。 *SQL-ID*_X.sql (X は数字。動的パラメーター照会時のみ) それ以外の場合は、指定した名前のファイル内に実行可能な SQL が出力されます。SQL-ID は各 SQL のコメントとして	必須ではない

	出力されます。デフォルト値は「.」です。	
useMarker	「true」または「false」を指定することができます。true を指定した場合は、ホスト変数がパラメーター・マーカー「?」に変換されます。false が指定された場合は変換されません。 DB2 の場合は Explain 時にパラメーター・マーカーを使用しなければならないため、true を指定してください。Oracle の場合は false を指定してください。デフォルト値は「true」です。	必須ではない
useExplainHeader	「true」または「false」を指定することができます。true を指定した場合、各 SQL の先頭に「EXPLAIN PLAN FOR」が付加されます。	必須ではない

表 7-13 コード生成タスクの属性

7.3.5. Explain結果のレポート

このAntタスクを使用することにより、複数のExplain結果をHTML形式のレポートとして表示することができます。まず、7.3.4実行可能SQLの生成によって生成したSQLを利用し、Explain結果を取得します。

```
(DB2 V9.5/Windows の場合)
@ECHO OFF
IF "DUMMY" == "DUMMY%4" (
   ECHO usage: db2explain.bat DBNAME DBUSER DBPASS DEST
) ELSE (

DATE /T
ECHO %3

FOR %%I IN (%4¥*.sql) DO db2expln -database %1 -u %2 %3 -stmtfile %%I -escape @ -output %%I.txt
-graph -opids -terminator ";"
)
@ECHO ON

(Oracle 11g/Windows の場合)
@ECHO OFF
IF "DUMMY" == "DUMMY%3" (
   ECHO usage: oracleexplain.bat DBUSER DBPASS DEST
) ELSE (

DATE /T
FOR %%I IN (%3¥*.sql) DO (
   TYPE %%I format.head > %%I.tmp
   sqlplus %1/%2 < %%I.tmp > %%I.txt
)
DEL %3¥*.tmp
)
@ECHO ON

(ファイル：format.head)
SET LINESIZE 1000;
SET PAGESIZE 10000;
select plan_table_output from table(dbms_xplan.display('plan_table',null,'serial'));
```

次に、Ant タスクによって Explain 結果のレポートを出力します。Ant プロジェクト(build.xml)の記述例は次の通りです。

```
<project name="ExplainReport" default="report">
   <!-- タスク定義 -->
   <taskdef name="explainReport"
         classname="com.ibm.jp.daogen.db.ant.ExplainReportTask"
         classpath="WACs-Commons.jar;base.jar;daogendb.jar" />

   <!—Explain 結果のレポート -->
   <target name="report">
```

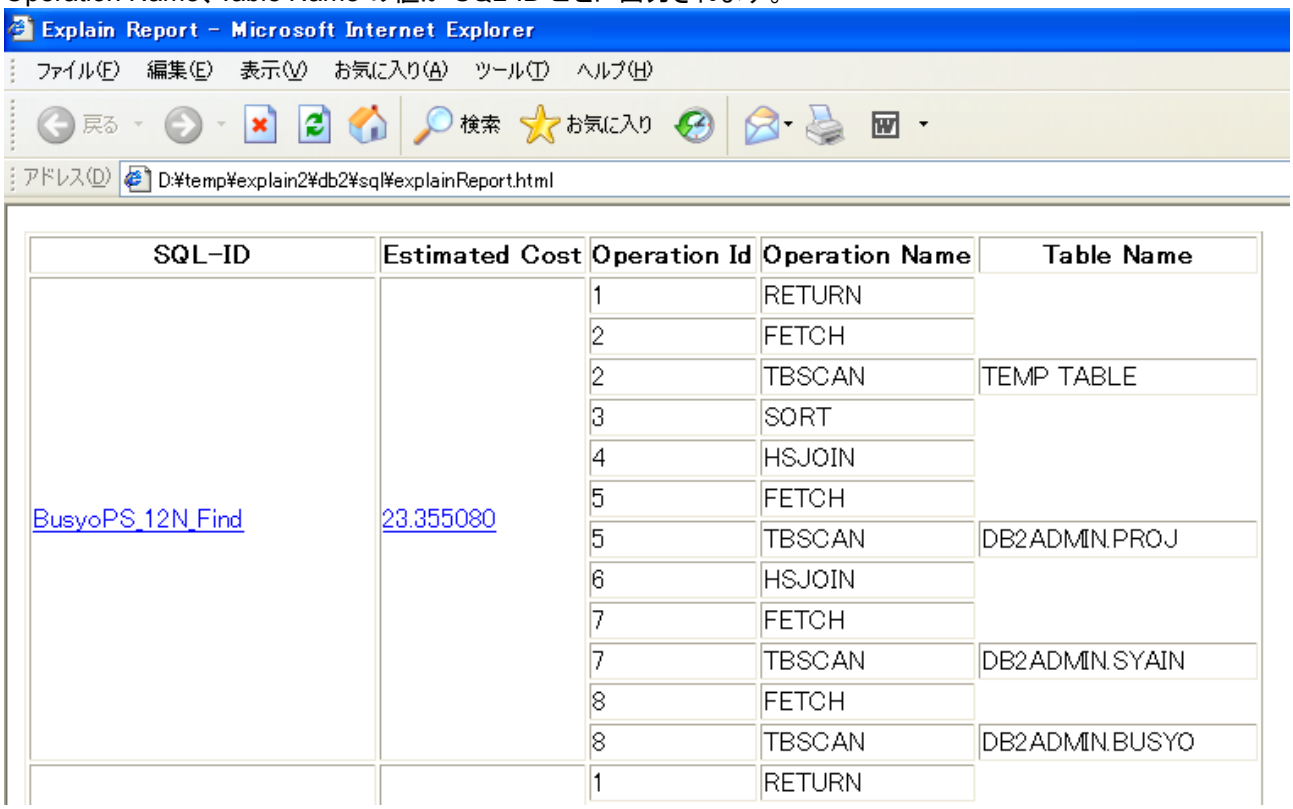
```
    <explainReport />
  </target>
</project>
```

<taskdef> タグの classname 属性に Explain 結果のレポート用のクラス「com.ibm.jp.daogen.db.ant.ExplainReportTask」を指定します。Explain 結果のレポート用タスク(図中<explainReport>タグ)に指定する属性の説明は次の通りです。

属性	説明	必須
dbms	「DB2」または「Oracle」を指定することができます。デフォルト値は「DB2」です。	必須ではない
path	Explain 結果のディレクトリー名を指定します。デフォルト値はカレント・ディレクトリーです。なお、Explain 結果ファイルの拡張子は「txt」である必要があります。	必須ではない
dest	レポートの出力先ディレクトリー名を指定します。デフォルト値はカレント・ディレクトリーです。	必須ではない

表 7-14 Explain 結果のレポート用タスクの属性

タスクが成功した場合、次のようなレポートが出力されます。Explain 結果に含まれる Estimated Cost、Operation Id、Operation Name、Table Name の値が SQL-ID ごとに出力されます。



SQL-ID	Estimated Cost	Operation Id	Operation Name	Table Name
BusyoPS_12N_Find	23.355080	1	RETURN	
		2	FETCH	
		2	TBSCAN	TEMP TABLE
		3	SORT	
		4	HSJOIN	
		5	FETCH	
		5	TBSCAN	DB2ADMIN.PROJ
		6	HSJOIN	
		7	FETCH	
		7	TBSCAN	DB2ADMIN.SYAIN
		8	FETCH	
		8	TBSCAN	DB2ADMIN.BUSYO
		1	RETURN	

図 7-17 Explain 結果レポート

7.3.6. Explain結果の比較レポート

このAntタスクを使用することにより、Explainの比較結果をHTML形式のレポートとして表示することができます。例えば索引付与前後のアクセスパスの変化を確認することができます。まず、7.3.4実行可能SQLの生成を参考にして、索引付与前と後のExplain結果を取得します。取得した結果はそれぞれ「sql_1 (付与前)」、「sql (付与後)」ディレクトリーに格納します。次に、AntタスクによってExplainの比較結果のレポートを出力します。Antプロジェクト(build.xml)の記述例は次の通りです。

```
<project name="ExplainDiffReport" default="reportDiff">
  <!-- タスク定義 -->
```

```
    <taskdef name="explainDiffReport"
          classname="com.ibm.jp.daogen.db.ant.ExplainDiffReportTask"
          classpath="WACs-Commons.jar;base.jar;daogendb.jar" />

    <!—Explain 比較結果のレポート -->
    <target name="reportDiff">
       <explainDiffReport />
    </target>
</project>
```

<taskdef> タグの classname 属性に Explain 比較結果のレポート用のクラス「com.ibm.jp.daogen.db.ant.ExplainDiffReportTask」を指定します。Explain 比較結果のレポート用タスク(図中<explainDiffReport>タグ)に指定する属性の説明は次の通りです。

属性	説明	必須
dbms	「DB2」または「Oracle」を指定することができます。デフォルト値は「DB2」です。	必須ではない
oldExplain	過去の Explain 結果を格納したディレクトリー名を指定します。デフォルト値は「sql_1」です。なお、Explain 結果ファイルの拡張子は「txt」である必要があります。	必須ではない
newExplain	現在の Explain 結果を格納したディレクトリー名を指定します。デフォルト値は「sql」です。なお、Explain 結果ファイルの拡張子は「txt」である必要があります。	必須ではない
dest	レポートの出力先ディレクトリー名を指定します。デフォルト値はカレント・ディレクトリーです。なお、出力されるファイル名と同名のファイルが存在する場合は、既存のファイル名を変更し、世代管理します。	必須ではない
generation	出力されるレポートの世代数です。デフォルト値は 1 です。	必須ではない

表 7-15 Explain 比較結果のレポート用タスクの属性

タスクが成功した場合、次のようなレポートが出力されます。Explain 結果に含まれる Estimated Cost、Operation Id、Operation Name、Table Name の値が新旧の SQL-ID ごとに出力されます。なお、アクセスパスが変化し、かつ、Estimated Cost が上昇したものは赤色、下降したものは青色で背景が強調されます。

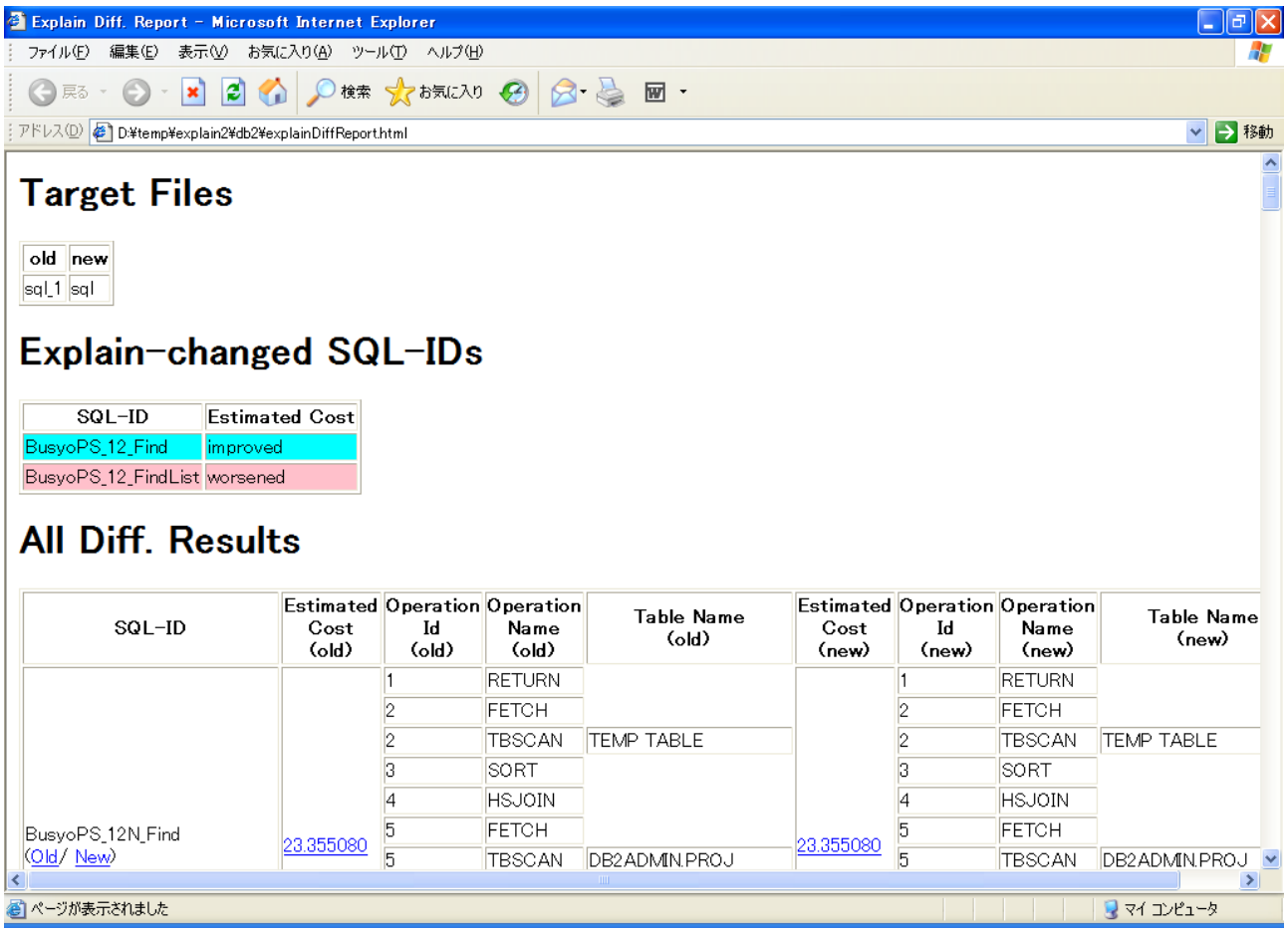


図 7-18 Explain 比較結果レポート

7.4. 制約事項

DAO 生成ツールには次の制約があります。

- ストアード・プロシージャーを使用する場合は、カスタム・メソッドとして独自に実装する必要があります。
- Oracle 上で SQLJ 実装を使用することはできません。
- DTO に primitive 型(int, short など)のフィールドを追加することはできません。
- n:n 関連を持つ表を、DTO の参照関係によって表現することはできません。
- 継承関係にある表を、DTO の継承関係によって表現することはできません。

7.5. バージョン間の差異について

過去の各バージョン間では、設定情報やクラスに次のような違いがあります。

v1.1 以前から v1.2

① デフォルト CRUD メソッドの引数について

v1.1 以前では、デフォルト CRUD メソッドの「select」の引数は「java.io.Serializable」でしたが、v1.2 以降では「Parameter」クラスとなっています。v1.1 の設定情報を v1.3 へ移行した場合は、従来どおり「java.io.Serializable」を引数に持つメソッドが出力されます。v1.3 上で新規に設定情報を作成した場合は、「Parameter」を引数に持つメソッドが出力されます。

v1.2 から v1.3

① 各設定情報の移行について

v1.2 以前で作成された「プロジェクト設定ファイル」を開くと、次のようなダイアログが表示されます。『OK』ボタンを押し、XML ファイルの移行を行なってください(移行に数分かかることがあります)。

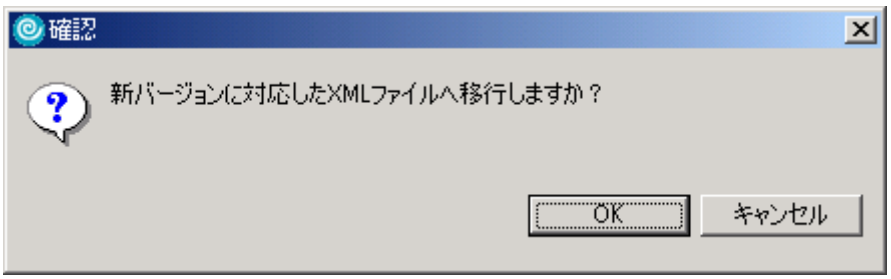


図 7-19 『確認』ダイアログ

② ExceptionHandler # handleException()のシグニチャー変更について
v1.2 以前の DAO 生成ツールでは、照会系・更新系ともに handleException を使用しています。
public void handleException(Exception e) throws DataAccessException
v1.3 では、以下の二種類の handleException を使用しています。
void handleQueryException(Exception e, int state) throws DataAccessException
void handleUpdateException(Exception e, int state) throws DataAccessException
v1.2 以前のプロジェクトを V1.3 に移行する場合は、ExceptionHandler.java を一度削除し、再度生成してください。

③ RowNotFoundException / MultiUpdatedException の変更について
DTO 取得照会/更新で、実行結果が 0 行であった場合に発生する RowNotFoundException と、取得結果が複数行だった場合に発生する RowMultiUpdatedException が、次のように変更になりました。

DTONotFoundException	DTO 取得照会で、実行結果が 0 件だった場合の例外
MultiDTOFoundException	DTO 取得照会で、実行結果が複数件だった場合の例外
DTONotUpdatedException	DTO 指定更新で、実行結果が 0 件だった場合の例外
MultiDTOUpdatedFoundException	DTO 指定更新で、実行結果が複数件だった場合の例外

また、挿入時に一意キーの重複エラーがあった場合に発生する例外『DuplicatedException』が追加されました。

v1.3 から v1.3.1

① Undefined メソッドについて
v1.3.0 までは、「INSERT」、「SELECT」、「UPDATE」、「DELETE」のいずれにも分類されないメソッドは「Undefined」として判定され、DAO インターフェース上にのみメソッドが追加されていました。v1.3.1 からは、このメソッドを廃止し、代わりに「カスタム・メソッド」として DAO インターフェース上へのメソッド追加を可能としています。

② JET 定義ファイルの書式について
JET 定義ファイルの詳細設定のうち、属性「output」に対する値が変更されました。変更は次のとおりです。
① 「yes」→「true」
② 「no」→「false」
③ 「override」→「overwrite」
JET 定義ファイルを変更していない場合は、次のファイルを削除してください。**削除せずにコード生成を行った場合、コードは出力されません。**
¥*「基準ディレクトリ」*¥Properties¥jet.properties
JET 定義ファイルを変更している場合は、上記の変更を JET 定義ファイルに適用してください。

v1.3.1 から v1.3.2

① *DAOName*DAO # findList*SQL-ID*(Parameter param, int rows)の扱いについて
*DAOName*DAO # findList*SQL-ID*(Parameter param, int rows)が非推奨となりました。
上記 API は非推奨となりました。代わりに findList*SQL-ID*(Parameter param, int begin, int rows)を使用して取得範囲を指定して下さい。第 2 引数は開始行番号(0～)です。第 2 引数に 0 を指定する事によって、findList*SQL-ID*(Parameter param, int rows)と同じ結果を得る事ができます。

v1.3.2 から v1.3.2.1

①*DAOName*SQLJDAO # connCtx - ConnectionContext フィールドについて
v1.3.2 までの実装では、sqlj.runtime.ConnectionContext # close()メソッドの呼び出しが行われていませんでした。
v1.3.2.1 では、各 CRUD メソッド内で ConnectionContext をインスタンス化し、close()メソッドの呼び出しを行うよう変更されています。 これに伴い上記フィールドが廃止されたため、*DAOName*CustomDAO 内で上記フィールドを参

照している箇所を削除し、 各メソッド内で ConnectionContext のインスタンス化および close 処理を行うように変更する必要があります。

②*DAOName*CustomDAO が持つ各フィールドについて
v1.3.2 までは、*DAOName*SQLJDAO と *DAOName*CustomDAO に次の protected フィールドが重複して宣言されていました。
conn - java.sql.Connection
exHandler - ExceptionHandler
connCtx - ConnContext
v1.3.2.1 では、上記の重複したフィールドが削除されました。 *DAOName*CustomDAO の各メソッド内でこれらのフィールドを参照している場合、コンパイル・エラーが発生します。 *DAOName*SQLJDAO のフィールドを参照するようエラー箇所を変更し、コンパイル・エラーを解消する必要があります。

v1.3.2.3 から v1.4

① 各設定情報の移行について
旧バージョンのプロジェクト・ファイル(.dgd ファイル)を右クリックし、「新バージョンへの移行（DAO Generator）」を選択してください。

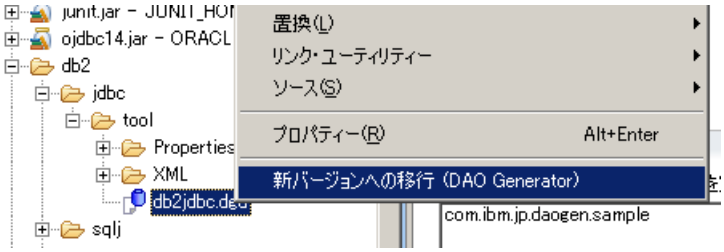


図 7-20 新バージョンへの移行

「移行が完了しました。」ダイアログが表示されれば、移行完了です。

図 7-21 移行完了ダイアログ

② ファイル・アクセス機能について
ファイル・アクセス機能が追加になりました。ファイル・アクセス機能とは、ファイルのヘッダー情報(DAO 生成ツール独自ファイル)からデータアクセス・コンポーネントを自動生成する機能です。具体的には、DAO 生成ツールは、テキストファイルからヘッダー情報を読み込み、DAO・DTOを生成します。業務ロジックからは、そのDAO・DTOを使用して、テキストファイルへのデータの読み込み・書き込みを行なうことができます。

③ クラス名について
次のクラスが変更されています。ただし、旧バージョンのプロジェクト基準ディレクトリー下に JET 定義ファイル(jet.properties)を含んだ状態で①の操作を行った場合は、以前のクラス名がそのまま移行されます。

*DAOName*DB2DAO/DAONameOracleDAO→*DAOName*JDBCDAO
*DAOName*Itrerators→*DAOName*Iterators

④ サンプル実装クラスについて
*DAOName*ClientSample の出力を廃止しました

v1.4 から v1.5

① DTO クラス名の変更
デフォルトの設定では、DTO クラス名がツール上の DTO 名と同一になりました(従来は「ツール上の DTO 名+DTO」)。ただし、v1.4 で作成したプロバイダー定義は v1.5 でも使用可能なため、v1.4 で作成したプロジェクトを使用した場合のクラス名は、従来どおり「ツール上の DTO 名+DTO」となります。

なお、v1.4 と v1.5 の間での各設定情報の移行は不要です。

v1.5 から v1.5.1
① Builder クラスの追加
「JDBC」実装を使用した場合、SQL 組み立て機能を持つ共通クラス「Builder」が新たに出力されるようになりました。同クラスは新たな動的パラメーター照会用変数「:_AND」、「:_OR」の組み立て機能を含んでいます。以前のバージョンで同クラスを使用するには、「StatementBuilder」および「PreparedStatementBuilder」クラスを一度削除し、再生成を行う必要があります(両クラスは「Builder」クラスのサブクラスとなります)。

② ExceptionHandler クラスの変更
コンストラクターを private から protected へ変更し、継承による例外処理を拡張可能としました。

③ DefaultLogger クラスの変更
「JDBC」実装、「SQLJ」実装を選択した場合に生成される DefaultLogger クラスに boolean を引数とするコンストラクターを追加し、true を指定した場合はログ出力時の SQL パラメーター値を「*」によってマスキングすることが可能となりました。

④カーソルが持つ Holdability の変更
「SQLJ」実装を選択した場合に出力される Iterator.sqlj クラスを変更し、更新可能カーソルの定義を「holdability=true」から「holdability=false」に変更しました。これにより、XA インターフェース上でカーソルを使用することが可能となりました。

なお、v1.4 と v1.5.1 の間での各設定情報の移行は不要です。

v1.5.1 から v2.0
主な変更点は次の通りです。

① Spring Framework 連携用 DAO の提供
Dependency Injection 技術の有力なオープンソース実装である Spring Framework と連携可能な DAO を生成できるようになりました。これにより、業務ロジックからの DAO 呼び出しが容易かつ柔軟になるとともに、Spring Framework が持つコネクション管理とトランザクション管理を利用できるようになりました。

② 共通抽象 DAO クラスの追加
自動生成される各 DAO クラスの共通親クラス(JDBCDAO/SQLJDAO/SpringDAO)を新たに設計しました。これにより、生成コードに含まれる冗長なロジックが共通化され、可読性向上、および、カスタム DAO クラスの実装効率が向上しました。

③ 非チェック例外への変更
生成される例外クラスのデフォルトの親クラスを java.lang.Exception から java.lang.RuntimeException に変更しました。例外クラス用/DAO クラス用のプロバイダーのプロパティーを変更することにより、これらの値は変更可能です。

④ DefaultLogger のデフォルト・パッケージの変更
生成される DefaultLogger のデフォルトのパッケージ名が、ルート・パッケージ(DAO と同一パッケージ)から、実装パッケージ(*.impl)に変更されました。なお、既存のプロバイダー・ファイルの設定が優先されるため、変更は新規にコード生成を実行する場合のみ有効です。

⑤ WACs 用 DAO の提供中止
v1.5.1 までは、生成される DAO クラスの実装として WACs (Web Application Components)の DB アクセス機能を利用したコードを生成していました。しかし、機能としては JDBC を利用した DAO クラス実装と同等であり、v2.0 では新たな Spring Framework との連携用 DAO のサポートが加わったことをきっかけとして、WACs 用 DAO のサポート

を中止しました。WACs 用 DAO クラスから v2.0 へバージョンアップする場合は、次の手順に沿って JDBC を利用した DAO クラスへ移行する必要があります。

1. Java プロジェクトのバックアップ
DAO 生成ツール用の定義ファイル群および生成されたコードが含まれる Java プロジェクトのバックアップを取得してください。

2. 実装パッケージの削除
実装パッケージ(デフォルトでは*.impl)を削除してください。

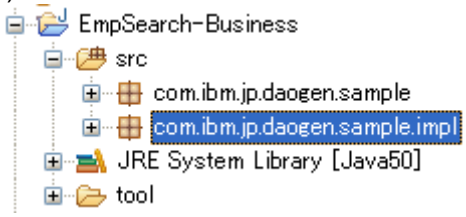


図 7-22 実装パッケージ

3. プロバイダー・グループ設定のリセット
Eclipse の「ナビゲーター」ビュー上から(「アウトライン」ビューでは削除対象のディレクトリーが表示されません)、「%基準ディレクトリー%/DB/Properties/.provider」下にある、「com.ibm.jp.daogen.codegen.db.wacs」ディレクトリーおよび「provider.properties」を削除してください。

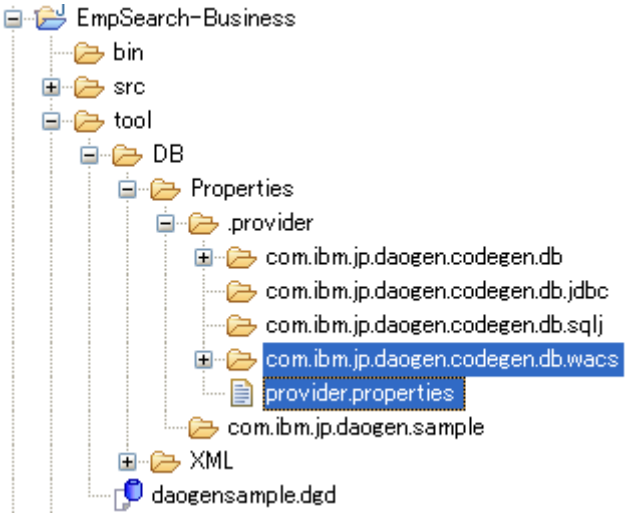


図 7-23 プロバイダー・グループ設定のリセット

4. プロバイダー・グループの再選択
プロジェクト定義ファイルを開き、5コード生成の指示に従ってプロバイダー・グループ「JDBC」を選択し、コードの再生成を行ってください。

なお、JDBC を利用した DAO クラスと WACs を利用した DAO クラスのインターフェースは同一であるため、呼び出し側から見た DAO クラスの互換性は維持されます。ただし、カスタム DAO を使用されている場合、親クラスが提供するメソッド・フィールドの相異により、コンパイル・エラーとなる可能性があります。もしカスタム DAO を使用されている場合は、DAO 生成ツールの提供元へご相談ください。

v2.0 から v2.0.1

① バッチ更新用 API の追加
java.sql.Statement # executeBatch()を利用して、複数の DTO を更新するための API が出力できるようになりました。使用する場合は JDBCDAO/SpringJDBCDAO/SQLJDAO クラスを一度削除し、再度生成してください。

v2.0.1 から v2.0.2

① java.sql.Statement # setQueryTimeout()の利用
Parameter クラスに setQueryTimeout() メソッドが追加されました。機能を有効とするため、

Parameter/JDBCDAO/SpringJDBCDAO/SQLJDAO クラスを一度削除し、再度生成してください。

v2.0.2 または v2.0.3 から v2.1

① SQLJ 実装の Spring サポート

SQLJ実装によるDAOをSpringによって他のクラスへインジェクションすることができるようになりました。Springの利用方法は「7.1 Springとの連携」をご覧ください。

② 実行可能 SQL ファイル作成機能

SQL定義ファイル(sql.properties)から実行可能なSQLファイルを作成するための機能がAntタスクとして追加されました。Antタスクの詳細は「7.3Ant」をご覧ください。

③ Oracle 11g でのバッチ例外対応

Oracle 11g では、PreparedStatement によるバッチ実行時(PreparedStatement # executeBatch)において、例外が発生した時点でバッチ処理が終了するようになりました。DAO 生成ツールでは、バッチ例外(BatchIncompleteException)発生時に更新に成功した DTO のリストと失敗した DTO のリストを取得することができますが、この変更を吸収するために ExceptionHandler # handleBatchException()を更新しました。機能を有効とするため、ExceptionHandler クラスを一度削除し、再度生成してください。

v2.1 から v2.2

① 各設定情報の移行について

旧バージョンのプロジェクト・ファイル(.dgd ファイル)を右クリックし、「新バージョンへの移行（DAO Generator）」を選択してください。

図 7-24 新バージョンへの移行

「移行が完了しました。」ダイアログが表示されれば、移行完了です。

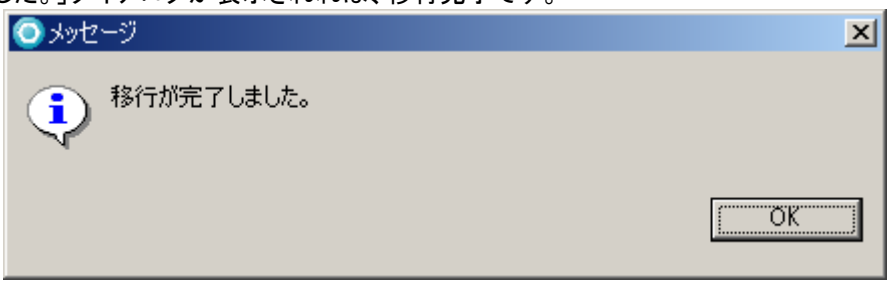


図 7-25 移行完了ダイアログ

旧バージョンのプロジェクト・ファイル(.dgd)と同一ディレクトリーに、新バージョンのプロジェクト・ファイル(.dgdx)が作成されています。旧バージョンのプロジェクト・ファイルを削除し、以降は新バージョンのプロジェクト・ファイルを使用してください。

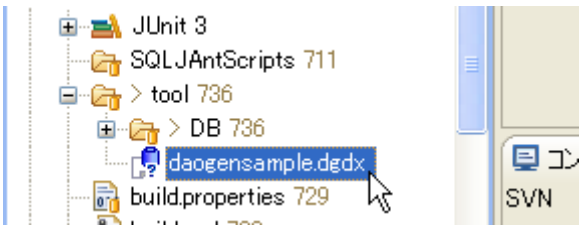


図 7-26 新バージョンのプロジェクト・ファイル(.dgdx)

② Spring 用 DAO への Logger インジェクションについて
@Autowired アノーテーションにより、Spring 用 DAO クラスに Logger インターフェースの実装クラスが自動的にインジェクションされるようになりました。このため、Bean 定義ファイルに Logger インターフェースの実装クラスを明示的に指定する必要があります。

```
<!-- DAO 用ログ情報 -->
        <bean id="logger" class="DefaultLogger クラス名" />
```

v7.0.4 から v7.0.5

① SQLJ 実装の更新可能イテレーターについて
生成される更新可能イテレーターの宣言文から、更新対象列の指定(updateColumn 属性)が省略されるようになりました。これは、更新 SQL のパラメーター名と DDL 上の列名が異なった場合に対応するためです。更新対象列の指定を生成するには、「IteratorProvider」の属性「useUpdateColumns」を「true」に指定してください。

v7.0.5 から v7.0.6

① SQL 定義ファイルの改行対応
『SQL 定義の編集』ダイアログで編集した SQL に改行が含まれていた場合、v7.0.5 までは改行が削除された SQL が SQL 定義ファイルに出力されていましたが、v7.0.6 では改行が削除されずに出力されるようになりました（ロード時に改行をエスケープするために行末にエスケープ文字が付加されます）。v7.0.5 までと同じように改行を削除する必要がある場合は、コード生成時に表示される『プロバイダー設定ウィザード』を用いて、以下の設定を行う必要があります。

1. プロバイダー・グループとして、利用しているツールが Base 版の場合は「JDBC」、Express 版の場合は「Spring(JDBC)」を選択。
2. プロバイダーとして「com.ibm.jp.daogen.codegen.db.jdbc.SqlLoaderProvider」を選択。
3. プロパティーで、key「keepLines」の values 値を「false」に設定。

② 根本例外を持たない DataAccessException への詳細メッセージ設定対応
DAO 生成ツールが独自に例外を生成するケースについて、例外に詳細メッセージが設定されるようになりました。

例外	詳細メッセージ
DTONotFoundException (JDBC のみ)	指定された条件に該当するレコードは存在しません。(SQLID={0} , Parameter={1})
MultiDTOFoundException (JDBC のみ)	指定された条件に該当するレコードが複数件存在します。(SQLID={0} , Parameter={1})
DTONoUpdatedException	更新対象が存在しません。(SQLID={0} , Parameter={1})
MultiDTOUpdatedException	複数件のレコードが更新/削除されました。(SQLID={0} , Parameter={1})

表 7-16 設定される詳細メッセージ

V7.0.5 までと同じように詳細メッセージを設定しないようにする必要がある場合には、コード生成時に表示される『プロバイダー設定ウィザード』を用いて、以下の設定を行う必要があります。

1. 各プロバイダー・グループに応じて以下のプロバイダーを選択
   - 「JDBC」の場合 ：com.ibm.jp.daogen.codegen.db.jdbc.JDBCDAOProvider
   - 「JDBC(Spring)」の場合 ：com.ibm.jp.daogen.codegen.db.spring.jdbc.SpringJDBCDAOProvider
   - 「SQLJ」の場合 ：com.ibm.jp.daogen.codegen.db.sqlj.SQLJDAOProvider
   - 「SQLJ(Spring)」の場合 ：com.ibm.jp.daogen.codegen.db.spring.sqlj.SpringSQLJDAOProvider
2. プロパティーで、key「useExceptionMessage」の values 値を「false」に設定。

v7.0.6 から v7.1.1

① 共通クラスの更新
機能追加に伴い、共通クラスにメソッドが追加されました。V7.1.1 以降で生成される DAO クラスは追加されたメソッドに依存しているため、JDBCDAO/SpringJDBCDAO/SQLJDAO クラスを一度削除し、再度生成してください。

② CHARACTER 型の取得結果の右トリムについて
V7.1.1 から CHARACTER 型のカラムの取得結果がデフォルトで右トリムされるようになりました。V7.0.6 以前の右ト

リムしない動作に戻す場合は、コード生成時に表示される『プロバイダー設定ウィザード』を用いて、以下の設定を行う必要があります。

1. 各プロバイダー・グループに応じて以下のプロバイダーを選択
   - 「JDBC」の場合 ：com.ibm.jp.daogen.codegen.db.jdbc.AbstractDAOProvider
   - 「JDBC(Spring)」の場合 ：com.ibm.jp.daogen.codegen.db.spring.jdbc. AbstractDAOProvider
   - 「SQLJ」の場合 ：com.ibm.jp.daogen.codegen.db.sqlj. AbstractDAOProvider
   - 「SQLJ(Spring)」の場合 ：com.ibm.jp.daogen.codegen.db.spring.sqlj. AbstractDAOProvider
2. プロパティーで、key「trim」の values 値を「false」に設定。

② DTO の getter/setter について

V7.1.1 から DTO の getter/setter について JavaBean 仕様に準拠したメソッド名で出力されるようになりました。V7.0.6 以前と同様のメソッド名としたい場合は、コード生成時に表示される『プロバイダー設定ウィザード』を用いて、以下の設定を行う必要があります。

3. 各プロバイダー・グループに応じて以下のプロバイダーを選択
   - 「JDBC」の場合 ：com.ibm.jp.daogen.codegen.db.jdbc.AbstractDAOProvider
   - 「JDBC(Spring)」の場合 ：com.ibm.jp.daogen.codegen.db.spring.jdbc. AbstractDAOProvider
   - 「SQLJ」の場合 ：com.ibm.jp.daogen.codegen.db.sqlj. AbstractDAOProvider
   - 「SQLJ(Spring)」の場合 ：com.ibm.jp.daogen.codegen.db.spring.sqlj. AbstractDAOProvider
4. プロパティーで、key「accessorNameStrict」の values 値を「false」に設定。
5. プロバイダーとして、「com.ibm.jp.daogen.codegen.db.DTOProvider」を選択
6. プロパティーで、key「accessorNameStrict」の values 値を「false」に設定。