Batch Framework for Java 3.5.0

チュートリアルマニュアル

第 1.0.0 版

株式会社 NTTデータ

■ 改訂履歴

版数	版改訂日	変更箇所	変　更　理　由	変　更　内　容
1.0.0	2015年2月	初版発行	－	－

TERASOLUNA Batch Framework for Java 3.5.0 チュートリアルマニュアル

第 1 章　概要 4
1.1　本資料の目的 4
1.2　バッチフレームワーク概略 4

第 2 章　バッチフレームワークチュートリアル 7
2.1　チュートリアル学習環境の整備 7
2.2　入力が DB、出力がファイルである場合のジョブ(同期型ジョブ実行) 15
2.2.1.　定義ファイルの作成 18
2.2.2.　ビジネスロジックの実装 19
2.2.3.　起動と確認 28
2.3　入力がファイル、出力が DB である場合のジョブ(同期型ジョブ実行) 30
2.3.1.　定義ファイルの作成 33
2.3.2.　ビジネスロジックの実装 34
2.3.3.　起動と確認 38
2.4　入力チェック機能を利用した、入力がファイル、出力が DB である場合のジョブ(同期型ジョブ実行) 39
2.4.1.　定義ファイルの作成 42
2.4.2.　ビジネスロジックの実装 43
2.4.3.　起動と確認 52
2.5　コントロールブレイク機能を利用した、入力がファイル、出力がファイルである場合のジョブ(同期型ジョブ実行) 53
2.5.1.　定義ファイルの作成 57
2.5.2.　ビジネスロジックの実装 57
2.5.3.　起動と確認 66
2.6　入力が DB、出力がファイルである場合のジョブ(非同期型ジョブ実行) 67
2.6.1.　定義ファイルの作成 70
2.6.2.　ビジネスロジックの実装 71
2.6.3.　起動と確認 75

第 3 章　APPENDIX 78
3.1　概要 78
3.2　チュートリアル学習環境の整備(Oracle) 78

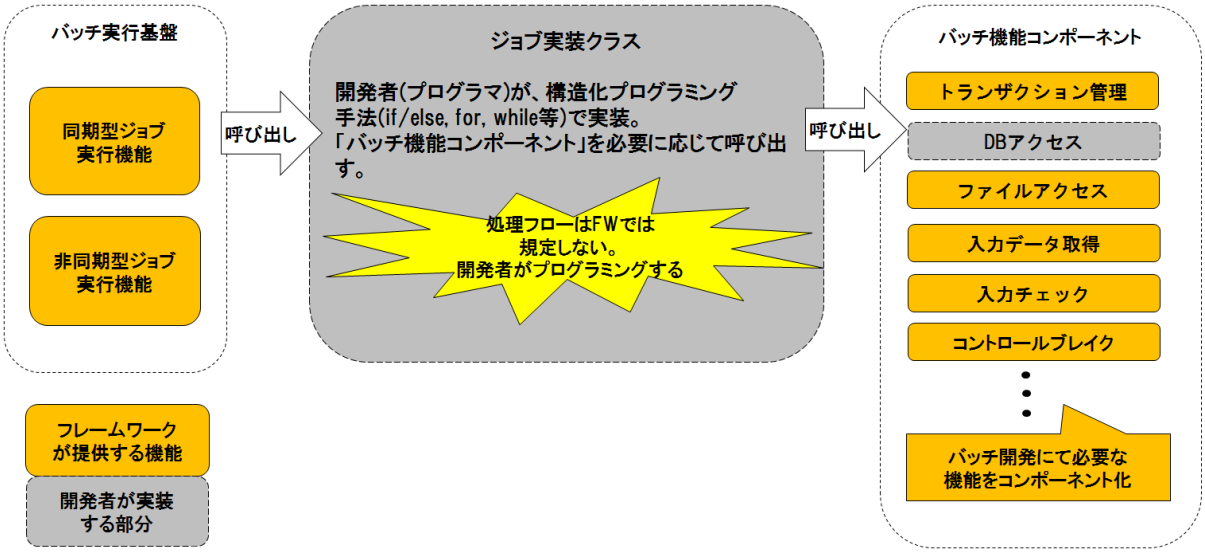
第1章　概要

1.1 本資料の目的

本資料の目的は説明に従って設定ファイル、プログラム等を作成していくことにより、TERASOLUNA Batch Framework for Java(以下、バッチフレームワークと呼ぶ)の基本的な処理パターンと実装方法を学習することである。

1.2 バッチフレームワーク概略

- バッチフレームワークのコンセプト
  - 必要な機能は、コンポーネント化することによって、実現したい処理を取捨選択しやすく、段階的学習が可能なアーキテクチャとしている(オンラインの開発者は、すぐにでもバッチ開発を習得することが可能)。



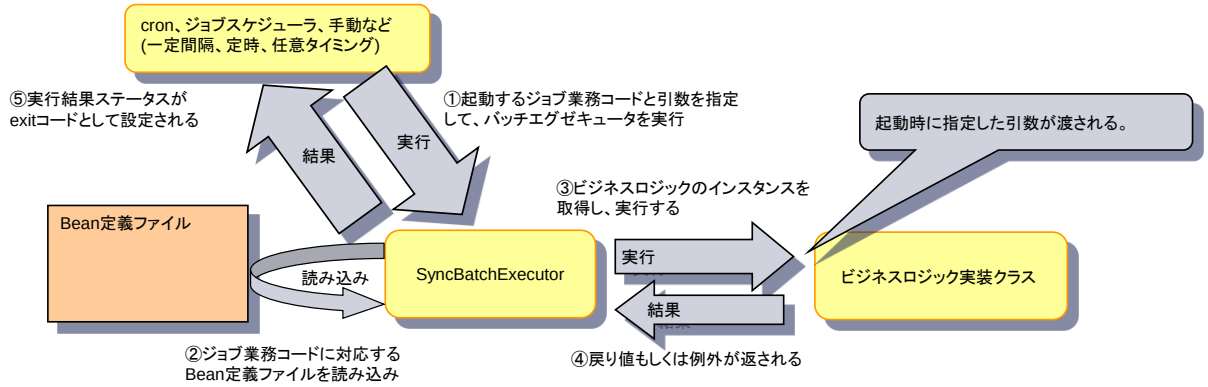
- コンポーネントの概要

  バッチフレームワークのコンポーネントの概要を以下に示す。

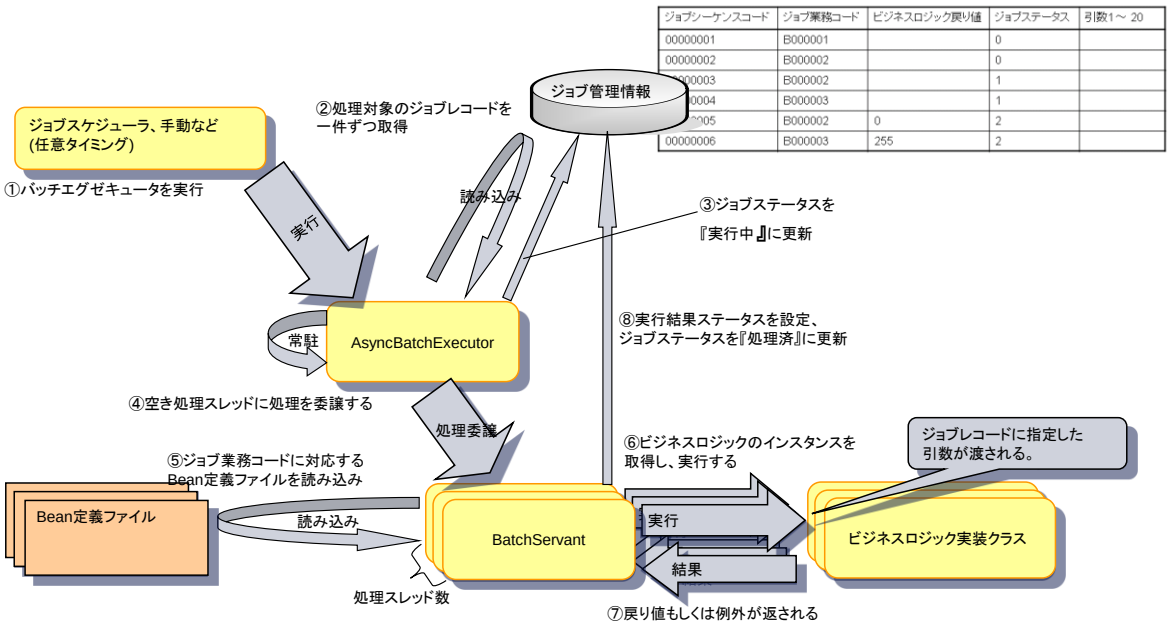
コンポーネント名	解説	チュートリアルでの使用
BL01 同期型ジョブ実行	新規にプロセスを起動して、ジョブを実行する機能	2 章の 2.2 節、2.3 節、2.4 節、2.5 節を参照
BL02 非同期型ジョブ実行	ジョブ管理テーブルに登録されたジョブ情報をもとに、スレッドにてジョブを実行する機能	2 章の 2.6 節を参照
BL03 トランザクション管理	コミット、ロールバックなどのユーティリティメソッドを提供する機能	2 章の 2.2 節、2.3 節、2.4 節、2.5 節、2.6 節を参照

コンポーネント名	解説	チュートリアルでの使用
BL04 例外ハンドリング	ビジネスロジック内で例外が発生した場合、発生した例外をハンドリングし、ジョブ終了コードを設定することができる機能	
BL05 ビジネスロジック実行	ビジネスロジックを実行するためのインタフェース、抽象クラスの提供	2 章の 2.2 節、2.3 節、2.4 節、2.5 節、2.6 節を参照
BL06 DB アクセス	MyBatis3 を利用した、O／R マッピング機能	2 章の 2.2 節、2.3 節、2.4 節、2.6 節を参照
BL07 ファイルアクセス	CSV や固定長ファイルを、オブジェクトにマッピングする機能	2 章の 2.2 節、2.3 節、2.4 節、2.5 節、2.6 節を参照
BL08 ファイル操作	ファイル名変更、ファイル移動、ファイルコピー、ファイル削除、ファイル結合をするための機能	
BL09 メッセージ管理	ログ出力等で利用する文字列を管理するための機能	
AL041 入力データ取得	データ収集を行うモジュールで、以下の特徴を持つ機能 ・データベースから大量のデータを取得する際に、メモリの使用量をフェッチサイズ分のみに抑えることができる ・MyBatis3 の ResultHandler を利用した場合と異なり、構造化プログラミング(while 文)にて実装できる ・引数として、DB アクセスやファイルアクセスで利用する DAO を渡す ・バックグラウンドで動作するため、スループットが向上する ・1 件前後の値も取得できるため、コントロールブレイクに対応できる	2 章の 2.2 節、2.3 節、2.4 節、2.5 節、2.6 節を参照
AL042 コントロールブレイク	コレクタを利用して、現在読んだデータと、次に読むデータで、キーが切り替わるのを判定する機能	2 章の 2.5 節を参照
AL043 入力チェック	コレクタを利用して取得したデータに対して入力チェックを実行する機能 BeanValidation を利用する	2 章の 2.4 節を参照

- フレームワーク構造(同期型ジョブ実行)
  シェルからプロセスとして、バッチジョブを起動する。シェル引数が、ビジネスロジック実装クラス(ジョブクラス)の実行メソッド引数に渡され、メソッドの戻り値が、そのまま終了コードとして、シェルに戻される。



- フレームワーク構造(非同期型ジョブ実行)
  シェルからプロセスとして、バッチジョブを起動する。「ジョブ管理テーブル」に登録された情報を元にして、スレッドとしてジョブを起動する。



第2章　バッチフレームワークチュートリアル

2.1 チュートリアル学習環境の整備

本節では、チュートリアルを学習するための環境整備について説明する。本チュートリアルで使用する資材は以下の通りである。

- terasoluna-batch-blank：アプリケーションの元となるブランクプロジェクト
- terasoluna-batch-tutorial：チュートリアルの完成版プロジェクト

本チュートリアルではブランクプロジェクトに変更を加えることによってアプリケーションを作成する。

※備考
チュートリアル学習環境の整備を Oracle で行いたい場合は、「3.2 チュートリアル学習環境の整備(Oracle)」を参照のこと。

- 想定環境
  - フレームワーク： TERASOLUNA Batch Framework for Java 3.5.0
  - OS：Microsoft Windows 7 Professional
  - JDK：JDK 1.7.0_xx(x はバージョン番号)
  - データベース：PostgreSQL 9.3.x for Windows
  - 総合開発環境：Eclipse SDK 3.7.x
    ※Pleiades All in One 日本語ディストリビューション版

- インストール/開発環境の整備
  1. アプリケーションの用意
     本資料で必要となるアプリケーションを用意する。
     - JDK 1.7.0_xx(x はバージョン番号)
       ※http://www.oracle.com/technetwork/jp/java/javase/downloads/jdk7-downloads-1880260.html
     - PostgreSQL 9.3.x for Windows
       ※http://www.enterprisedb.com/products-services-training/pgdownload#windows
     - Eclipse SDK 3.7.x + Maven プラグイン
       ※http://mergedoc.sourceforge.jp/pleiades_distros3.7.html

2. アプリケーションのインストール

用意した各アプリケーションをインストールする。本資料ではアプリケーションをそれぞれ以下のディレクトリにインストールすると想定して記述している。

アプリケーション	インストールディレクトリ
JDK 1.7.0_xx(x はバージョン番号	C:¥Program Files¥Java¥jdk1.7.0_xx (x はバージョン番号)
Eclipse SDK 3.7.x	C:¥Eclipse

2.1 Java ホームディレクトリの設定

ここでは以下のように環境変数を設定する。

- JAVA_HOME：C:¥Program Files¥Java¥jdk1.7.0_xx (x はバージョン番号)
- path：%JAVA_HOME%¥bin;

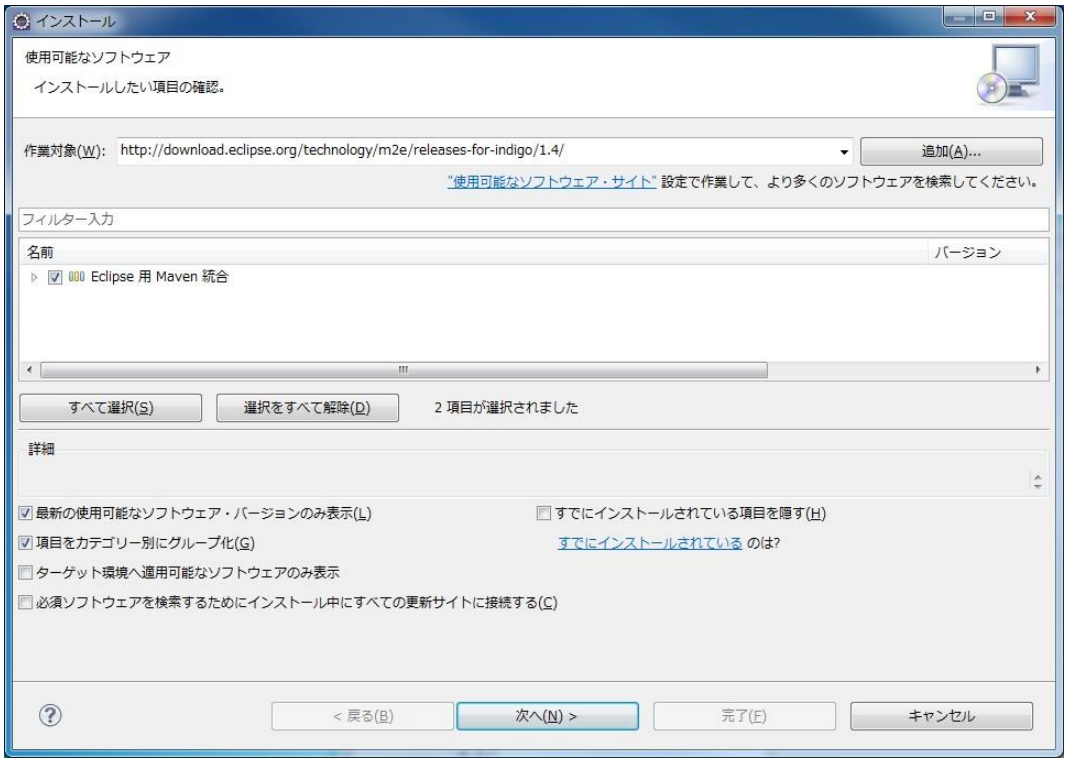
2.2 Eclipse の設定

インストール済み JRE の設定

- Eclipse の「ウィンドウ」→「設定」メニューを選択する。
- 「Java」→「インストール済みの JRE」で、「jdk1.7.0_xx (x はバージョン番号)」にチェックがついていることを確認する。
- 続けて「Java」→「コンパイラー」で、「JDK 準拠」のコンパイラー準拠レベルが“1.7”になるように設定する。

Maven プラグインの設定(本書では m2e を使用)

- Eclipse の「ヘルプ」→「新規ソフトウェアのインストール」メニューを選択する。
- 「作業対象」にダウンロード用の URL 「http://download.eclipse.org/technology/m2e/releases-for-indigo/1.4/」を入力すると、リストが更新されるので、「Eclipse 用 Maven 統合」のチェックボックスをつけ、「次へ」を選択する。



- インストール詳細を確認し、「次へ」を選択する。
- 表示されたライセンスを確認し、「使用条件の条項に同意します」のラジオボタンをチェックし、「完了」を選択する。
- 「ソフトウェア更新」ウインドウが出た場合は、「今すぐ再起動」を選択する。

✓ Maven 3.2.3 以降を使用する際の注意

上記方法でインストールした m2e に同梱される Maven のバージョンは 3.0.4 であるため該当しないが、Maven のバージョンが 3.2.3 以降の場合、セントラルリポジトリには HTTPS を使用してアクセスするため、「(ユーザーのホームディレクトリ)/.m2/settings.xml」に設定を追加する必要がある。追加する設定内容については、以下の URL を参照のこと。

「http://central.sonatype.org/pages/consumers.html#apache-maven」

ファイルが存在しない場合は、上述した URL の設定内容で、「settings.xml」を新しく作成すること。

✓ m2e 1.5.0 を使用する場合の注意

新しい Eclipse を使用する場合、m2e のバージョンが 1.5.0 となる場合がある。その場合は、m2e-apt プラグインのインストールが必要になる。Eclipse の「ヘルプ」→「Eclipse マーケットプレース」を選択し、検索テキストボックスに「m2e-apt」と入力すると、利用可能な m2e-apt プラグインを検索できる。

リストに表れた「m2e-apt」プラグインの「インストール」を押下し、プラグインのインストールを行うこと。

✓ ビルドができない場合の補足

これまでの設定でビルドができない場合、以下の設定を「(ユーザーのホームディレクトリ)/.m2/settings.xml」に追加し、HTTP でアクセスすること。

```
<settings>
    <mirrors>
        <mirror>
            <id>central</id>
            <mirrorOf>central</mirrorOf>
            <name>Central</name>
            <url>http://repo1.maven.org/maven2</url>
        </mirror>
    </mirrors>
</settings>
```

3. プロジェクトの準備

本資料で使用するチュートリアルアプリケーションではバッチフレームワークで提供しているブランクプロジェクトを使用する。ここではワークスペースを“C:¥Eclipse¥workspace”とし、“terasoluna-batch4j-blank_(バージョン番号).zip”ファイルをワークスペース直下に展開する。

4. プロジェクトのインポート

「3 プロジェクトの準備」で作成した“C:¥Eclipse¥workspace¥terasoluna-batch-blank”を Eclipse で編集できるようにインポートする。

(1). Eclipse を起動する。
(2).「ファイル」→「インポート」を選択する。
(3). 選択画面では「Maven＞Existing Maven Projects」を選択して、「次へ」を押下する。
(4).「Import Maven Projects」画面ではルートディレクトリの選択欄に“C:¥Eclipse¥workspace¥terasoluna-batch-blank”を指定し、「/pom.xml xxxxxx.yyyyyy.zzzzzz:terasoluna-batch-blank:(バージョン番号).jar」にチェックが入っていることを確認して、「完了」を押下する。
(5). パッケージエクスプローラーの“terasoluna-batch-blank”を右クリックし、「リファクタリング」→「名前の変更」を選択し、「新しい名前」のテキストボックスに“tutorial”と入力し「OK」を押下する。
(6). パッケージエクスプローラーの“pom.xml”選択し、アーティファクト Id のテキストボックスに“tutorial”と入力し、保存する。

(7). パッケージエクスプローラーの“tutorial”を右クリックし、「Maven＞プロジェクトの更新」を選択し、“tutorial”にチェックがついていることを確認し、「OK」を押下する。

5. データベースの設定

(1). PostgreSQL9.3.x のインストールを行う。
本チュートリアルでは以下の設定のようにインストールしたと仮定する。

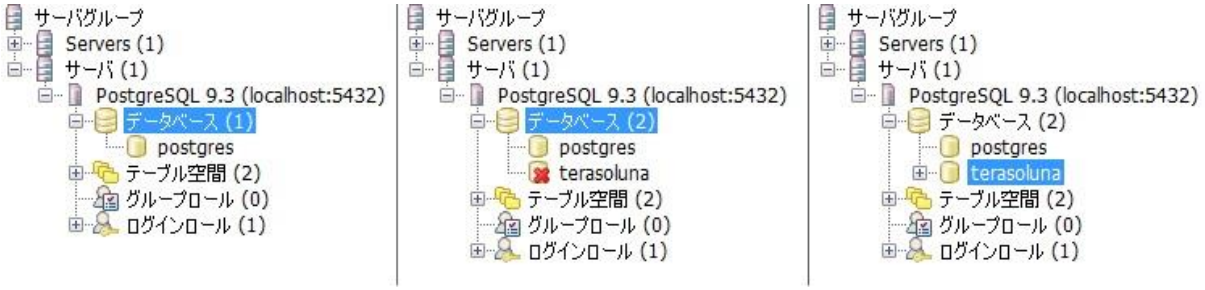
- ディレクトリ：C:¥Program Files¥PostgreSQL¥9.3 (デフォルト)
- ID：postgres
- Password：P0stgres [ﾋﾟｰｾﾞﾛｴｽﾃｨｰｼﾞｰｱｰﾙｲｰｴｽ]

(2). Windows のスタートメニューから[プログラム]→[PostgreSQL 9.3]→[pgAdmin III]を起動する。
「PostgreSQL 9.3(localhost:5432)」を選択し、右クリックメニューから「接続」を選択する。パスワード入力ウィンドウが表示されたら、「P0stgres [ﾋﾟｰｾﾞﾛｴｽﾃｨｰｼﾞｰｱｰﾙｲｰｴｽ]」と入力する。

(3). 画面左のメニューから「データベース(1)」を選択し、右クリックメニューより「新しいデータベース」を選択し、以下の内容を入力し「OK」を押下する。

- 名前：terasoluna
- エンコーディング :UTF8

(4). 画面左のメニューから「データベース(2)」「terasoluna」を選択すると、×印が消えてデータベースの内容が表示されることを確認する。



6. 入力ファイルの取得

「2.3 入力がファイル、出力が DB である場合のジョブ(同期型ジョブ実行)」および「2.4 入力チェック機能を利用した、入力がファイル、出力が DB である場合のジョブ(同期型ジョブ実行)」、「2.5 コントロールブレイク機能を利用した、入力がファイル、出力がファイルである場合のジョブ(同期型ジョブ実行)」で使用するファイルを以下の手順で取得する。

(1). パッケージエクスプローラーで、“tutorial”プロジェクトを右クリックする。

(2). 「新規」→「フォルダー」を選択し、フォルダー名に“inputFile”と入力し「完了」を押下する。

(3). チュートリアル完成版プロジェクト“terasoluna-batch-tutorial¥inputFile”フォルダー内にある“SMP002_input.csv”、“SMP003_input.csv”、“SMP004_input.csv”を(2)で作成した“inputFile”の直下にコピーする。

7. ファイルの出力先の作成

「2.2 入力が DB、出力がファイルである場合のジョブ(同期型ジョブ実行)」および「2.5 コントロールブレイク機能を利用した、入力がファイル、出力がファイルである場合のジョブ(同期型ジョブ実行)」、「2.6 入力が DB、出力がファイルである場合のジョブ(非同期型ジョブ実行)」で作成されるファイルの出力先を以下の手順で作成する。

(1). “tutorial”プロジェクトを右クリックする。
(2). 「新規」→「フォルダー」を選択し、フォルダー名に“outputFile”と入力し「完了」を押下する。

8. “jdbc.properties”の修正

本チュートリアルでは PostgreSQL を使用するため、
“tutorial¥src¥main¥resources¥mybatisAdmin”フォルダーと
“tutorial¥src¥main¥resources¥mybatis”フォルダーにある“jdbc.properties”を以下のように書き換える。

```
#ドライバー
jdbc.driver=org.postgresql.Driver
#URL
jdbc.url=jdbc:postgresql://127.0.0.1:5432/terasoluna
#ユーザー名
jdbc.username=postgres
#パスワード
jdbc.password=P0stgres
```

ﾋﾟｰｾﾞﾛｴｽﾃｨｰｼﾞｰｱｰﾙｲｰｴｽ

9. “AdminDataSource.xml”の修正

本チュートリアルでは PostgreSQL を使用するため、
“tutorial¥src¥main¥resources¥beansAdminDef”フォルダーにある
“AdminDataSource.xml”のシステム利用 DAO 定義(Oracle)がコメントアウトされていて、システム利用 DAO 定義(PostgreSQL)が有効になっていることを確認する。

```
    <!-- システム利用DAO定義(Oracle)
    <bean id="systemDao" class="org.mybatis.spring.mapper.MapperFactoryBean">
        <property name="mapperInterface"
            value="jp.terasoluna.fw.batch.executor.dao.SystemOracleDao"/>
        <property name="sqlSessionFactory" ref="sysSqlSessionFactory"/>
    </bean>
    -->
    <!-- システム利用DAO定義(PostgreSQL) -->
    <bean id="systemDao" class="org.mybatis.spring.mapper.MapperFactoryBean">
        <property name="mapperInterface"
            value="jp.terasoluna.fw.batch.executor.dao.SystemPostgreSQLDao"/>
        <property name="sqlSessionFactory" ref="sysSqlSessionFactory"/>
    </bean>
```

- データベースの初期化手順
  チュートリアル完成版プロジェクト“terasoluna-batch-tutorial”では以下のデータベースの初期化スクリプトとバッチファイルを提供する。

スクリプト／バッチファイル	説明
terasoluna-batch-tutorial¥sql¥postgres¥create_sequence_job_control.sql	チュートリアルで使用するジョブ管理シーケンスを作成する。
terasoluna-batch-tutorial¥sql¥postgres¥create_table_job_control.sql	チュートリアルで使用するジョブ管理テーブルを作成する。初期値は挿入されない。
terasoluna-batch-tutorial¥sql¥postgres¥create_table_nyusyukkin.sql	チュートリアルで使用する入出金テーブルを作成する。初期値は挿入されない。
terasoluna-batch-tutorial¥sql¥postgres¥drop_all_sequence.sql	チュートリアルで使用するジョブ管理シーケンスを削除する。
terasoluna-batch-tutorial¥sql¥postgres¥drop_all_tables.sql	チュートリアルで使用する全テーブルを削除する。
terasoluna-batch-tutorial¥sql¥postgres¥insert_all_data.sql	チュートリアルで使用するテーブルに初期値を挿入する。
terasoluna-batch-tutorial¥sql¥postgres¥terasoluna_tutorial_batch.sql	上記の SQL をすべて実行し、チュートリアルアプリケーションを実行可能な初期状態を作成する。
terasoluna-batch-tutorial¥sql¥postgres¥setup_for_PostgreSQL.bat	“terasoluna_tutorial_batch.sql”を実行するバッチファイル。

- “setup_for_PostgreSQL.bat”を実行してデータベースをチュートリアルアプリケーション実行可能な初期状態とする。

- JDBC のクラスパス設定

  Eclipse から PostgreSQL の JDBC へのクラスパス設定を行う。

  (1). PostgreSQL の JDBC “postgresql-9.3-x.jdbc41.jar” を Web サイト “http://jdbc.postgresql.org/download.html” から任意の場所にダウンロードする。

  (2). “tutorial” を右クリックし、「プロパティー」を選択する。

  (3). 「Java のビルド・パス」を選択し、「ライブラリー」タブを押下する。

  (4). 「外部 JAR の追加」を押下しダウンロードした “postgresql-9.3-x.jdbc41.jar” を選択する。

  (5). 「OK」を押下する。

2.2 入力が DB、出力がファイルである場合のジョブ（同期型ジョブ実行）

- 必要な作業

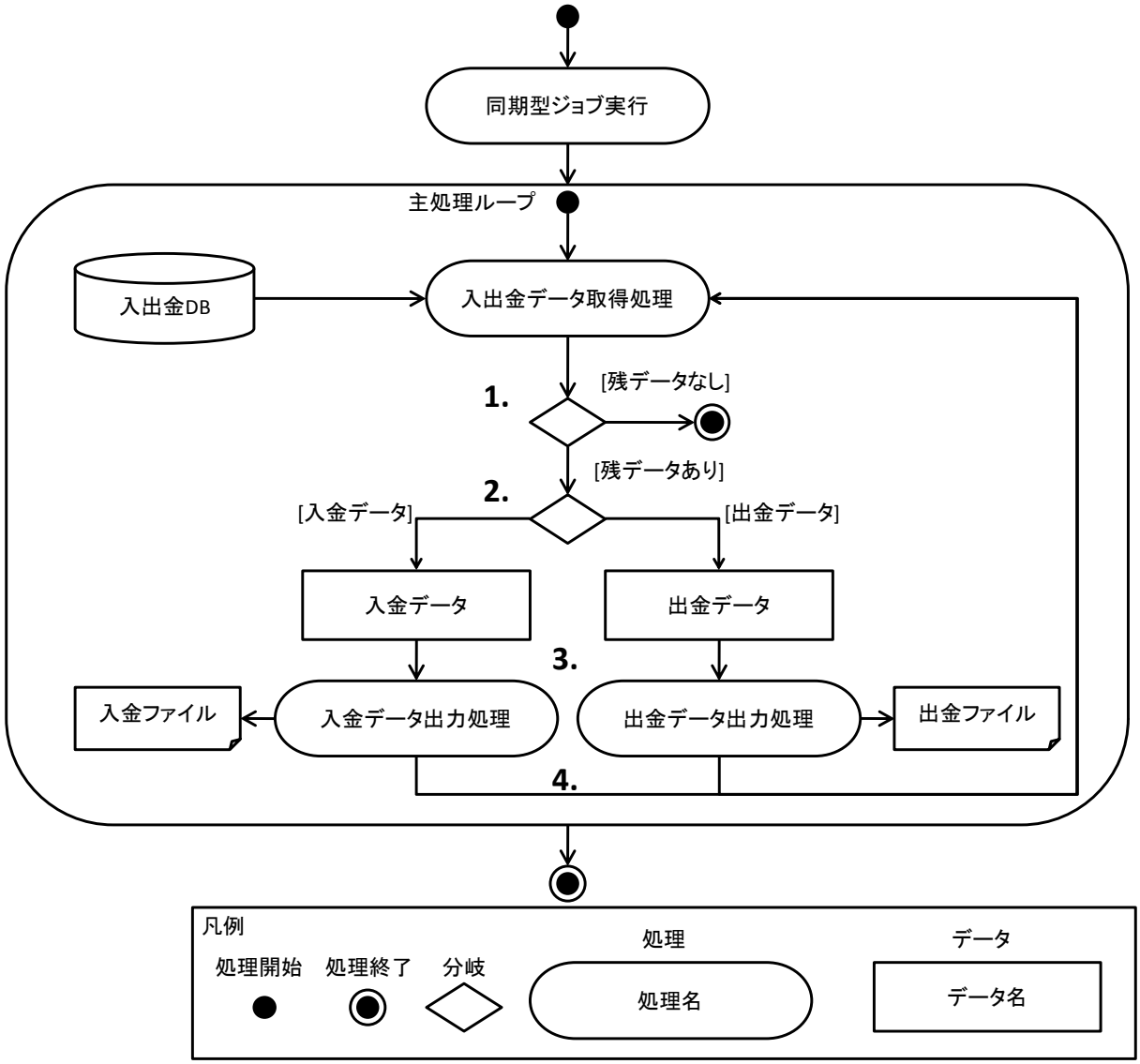
  2.2.1 定義ファイルの作成

  2.2.2 ビジネスロジックの実装

  2.2.3 起動と確認

- ジョブの内容

  図のように DB から顧客毎の入出金データを読み込み、入金データと出金データに分けてファイル出力を行うジョブを作成する。なお、本節では同期型ジョブ実行を採用する。



1. データを取得し、存在しない場合は処理を終了する。
2. 入力パラメータの「入出金区分」を確認し、入金データと出金データに分割する。

3. 入金データは入金ファイル、出金データは出金ファイルへ出力する。
4. 1.へ戻り、次の入出金データを取得する。

- ジョブのリソース

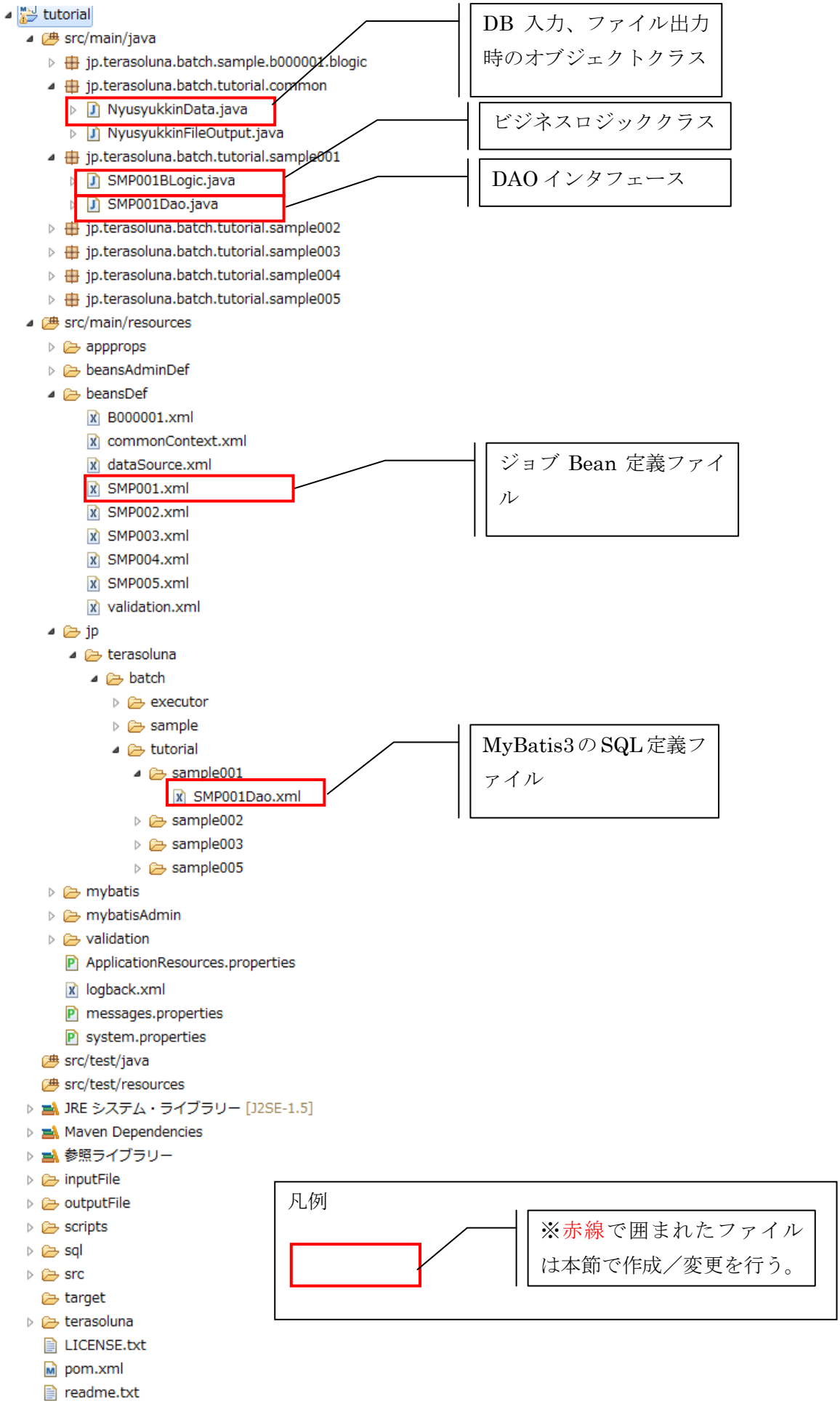
  本ジョブで作成／使用するパッケージ、ジョブ ID は以下の通りである。

  パッケージ：jp.terasoluna.batch.tutorial.common

  jp.terasoluna.batch.tutorial.sample001

  ジョブ ID： SMP001

  また、本ジョブ作成後のプロジェクトは以下のようになる。


tutorial
src/main/java
jp.terasoluna.batch.sample.b000001.blogic
jp.terasoluna.batch.tutorial.common
NyusyukkinData.java
NyusyukkinFileOutput.java
jp.terasoluna.batch.tutorial.sample001
SMP001BLogic.java
SMP001Dao.java
jp.terasoluna.batch.tutorial.sample002
jp.terasoluna.batch.tutorial.sample003
jp.terasoluna.batch.tutorial.sample004
jp.terasoluna.batch.tutorial.sample005
src/main/resources
appprops
beansAdminDef
beansDef
B000001.xml
commonContext.xml
dataSource.xml
SMP001.xml
SMP002.xml
SMP003.xml
SMP004.xml
SMP005.xml
validation.xml
jp
terasoluna
batch
executor
sample
tutorial
sample001
SMP001Dao.xml
sample002
sample003
sample005
mybatis
mybatisAdmin
validation
ApplicationResources.properties
logback.xml
messages.properties
system.properties
src/test/java
src/test/resources
JRE システム・ライブラリー [J2SE-1.5]
Maven Dependencies
参照ライブラリー
inputFile
outputFile
scripts
sql
src
target
terasoluna
LICENSE.txt
pom.xml
readme.txt
DB 入力、ファイル出力時のオブジェクトクラス
ビジネスロジッククラス
DAO インタフェース
ジョブ Bean 定義ファイル
MyBatis3 の SQL 定義ファイル
凡例
※赤線で囲まれたファイルは本節で作成／変更を行う。

2.2.1. 定義ファイルの作成

- ジョブ Bean 定義ファイル
- マッピングファイル

1. ジョブ Bean 定義ファイル

   [手順]

   (1). “SMP001.xml” を作成

   i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources¥beansDef” フォルダーを右クリックする。

   ii. 「新規」→「ファイル」を選択し、ファイル名に “SMP001.xml” と入力し「完了」を押下する。

   iii. “SMP001.xml” に “tutorial¥src¥main¥resources¥beansDef¥B000001.xml” の内容をコピーし、以下のように変更する。

変更前

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.sample.b000001" />
```

変更後

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.tutorial.sample001" />

<!-- SMP001Dao設定 -->
<bean class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"
        value="jp.terasoluna.batch.tutorial.sample001.SMP001Dao" />
    <property name="sqlSessionTemplate" ref="sqlSessionTemplate" />
</bean>
```

2. マッピングファイル

   MyBatis3 の定義ファイルであり、SQL 定義情報を記述する。

   [手順]

   (1). “SMP001Dao.xml” を作成

   i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources” フォルダーを右クリックする。

   ii. 「新規」→「ファイル」を選択し、「親フォルダーを入力または選択」に “tutorial/src/main/resources/jp/terasoluna/batch/tutorial/sample001”、ファイル名に “SMP001Dao.xml” と入力し「完了」を押下する。

   iii. “SMP001Dao.xml” を以下のように編集する。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org/DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="jp.terasoluna.batch.tutorial.sample001.SMP001Dao">
    <!-- データ取得 -->
    <!-- 入出金テーブル -->
    <select id="collectNyusyukkinData"
        resultType="jp.terasoluna.batch.tutorial.common.NyusyukkinData">
        SELECT
        SHITENNAME AS "shitenName"
        ,KOKYAKUID AS "kokyakuId"
        ,NYUSYUKKINKUBUN AS "nyusyukkinKubun"
        ,KINGAKU AS "kingaku"
        ,TORIHIKIBI AS "torihikibi"
        FROM
        NYUSYUKKINTBL
    </select>
</mapper>
```

2.2.2. ビジネスロジックの実装

本節では、ビジネスロジックの実装方法について説明する。

[手順]

(1). “NyusyukkinData.java”を作成

SQL の結果を格納するクラスを作成する。なお、このクラスは“入出金テーブル”のレコード取得時には結果取得オブジェクトクラス、またファイル入出力時にはファイル行オブジェクトクラスとなる。そのため、ファイルの個々の項目の定義情報をアノテーションにより設定している。

※以降のサンプル作成においても本クラスを使用する。

i. パッケージエクスプローラーで“tutorial”を右クリックする。

ii. 「新規」→「パッケージ」を選択し、名前に“jp.terasoluna.batch.tutorial.common”を入力し、「完了」を押下する。

iii. パッケージエクスプローラーで、“jp.terasoluna.batch.tutorial.common”パッケージを右クリックする。

iv. 「新規」→「クラス」を選択し、ファイル名に“NyusyukkinData”と入力し「完了」を押下する。

v. “NyusyukkinData.java”を以下のように編集する。

```java
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.common;

import java.util.Date;

import jp.terasoluna.fw.file.annotation.FileFormat;
import jp.terasoluna.fw.file.annotation.InputFileColumn;
import jp.terasoluna.fw.file.annotation.OutputFileColumn;

/**
 * 入出金情報のパラメータクラス。
 */
@FileFormat(overWriteFlg = true, fileEncoding = "MS932")
public class NyusyukkinData {

    /**
     * 支店名
     */
    @InputFileColumn(columnIndex = 0)
    @OutputFileColumn(columnIndex = 0)
    private String shitenName;

    /**
     * 顧客ID
     */
    @InputFileColumn(columnIndex = 1)
    @OutputFileColumn(columnIndex = 1)
    private String kokyakuId;

    /**
     * 入出金区分 0:出金 1:入金
     */
    @InputFileColumn(columnIndex = 2)
    @OutputFileColumn(columnIndex = 2)
    private int nyusyukkinKubun;
```

ファイル入出力時に使用するアノテーション。
上書き可否、エンコーディングを設定する

ファイル入出力時に使用するアノテーション。
行オブジェクトのカラム番号を設定する。

```
/**
 * 取引金額
 */
@InputFileColumn(columnIndex = 3)
@OutputFileColumn(columnIndex = 3)
private int kingaku;

/**
 * 取引日
 */
@InputFileColumn(columnIndex = 4, columnFormat = "yyyyMMdd")
@OutputFileColumn(columnIndex = 4, columnFormat = "yyyyMMdd")
private Date torihikibi;

/**
 * 支店名を取得する。
 *
 * @return shitenName
 */
public String getShitenName() {
    return shitenName;
}

/**
 * 支店名を設定する。
 *
 * @param shitenName
 */
public void setShitenName(String shitenName) {
    this.shitenName = shitenName;
}

/**
 * 顧客IDを取得する。
 *
 * @return kokyakuId
 */
```

```
    public String getKokyakuId() {
        return kokyakuId;
    }

    /**
     * 顧客IDを設定する。
     *
     * @param kokyakuId
     */
    public void setKokyakuId(String kokyakuId) {
        this.kokyakuId = kokyakuId;
    }

    /**
     * 入出金区分を取得する。
     *
     * @return nyusyukkinKubun
     */
    public int getNyusyukkinKubun() {
        return nyusyukkinKubun;
    }

    /**
     * 入出金区分を設定する。
     *
     * @param nyusyukkinKubun
     */
    public void setNyusyukkinKubun(int nyusyukkinKubun) {
        this.nyusyukkinKubun = nyusyukkinKubun;
    }

    /**
     * 取引金額を取得する。
     *
     * @return kingaku
     */
    public int getKingaku() {
        return kingaku;
```

```
    }

    /**
     * 取引金額を設定する。
     *
     * @param kingaku
     */
    public void setKingaku(int kingaku) {
        this.kingaku = kingaku;
    }

    /**
     * 取引日を取得する。
     *
     * @return torihikibi
     */
    public Date getTorihikibi() {
        return torihikibi;
    }

    /**
     * 取引日を設定する。
     *
     * @param torihikibi
     */
    public void setTorihikibi(Date torihikibi) {
        this.torihikibi = torihikibi;
    }
}
```

(2). “SMP001Dao.java”を作成

DAO を作成する。

i. パッケージエクスプローラーで“tutorial”を右クリックする。

ii. 「新規」→「パッケージ」を選択し、名前に“jp.terasoluna.batch.tutorial.sample001”を入力し、「完了」を押下する。

iii. パッケージエクスプローラーで、“jp.terasoluna.batch.tutorial.sample001”パッケージを右クリックする。

iv. 「新規」→「インターフェース」を選択し、名前に“SMP001Dao”を入力し、「完了」を押下する。

v. “SMP001Dao.java”を以下のように作成する。

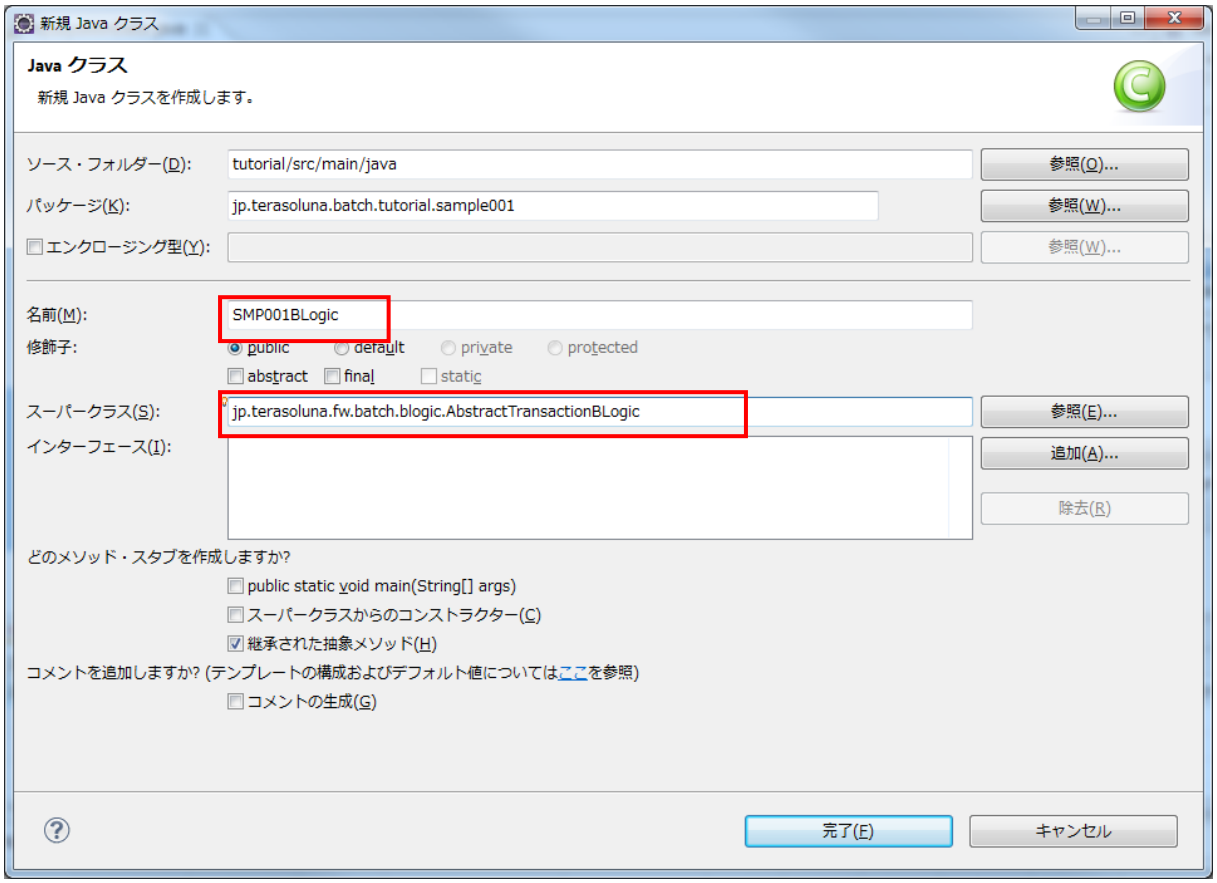
```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample001;

import org.apache.ibatis.session.ResultHandler;

public interface SMP001Dao {

    /**
     * 入出金情報を取得する。
     * @param object SQLパラメータ引数オブジェクト
     * @param handler ResultHandler
     */
    public void collectNyusyukkinData(Object object, ResultHandler handler);
}
```

(3). “SMP001BLogic.java”を作成

ビジネスロジックを作成する。

i. パッケージエクスプローラーで、“jp.terasoluna.batch.tutorial.sample001”パッケージを右クリックする。

ii. 「新規」→「クラス」を選択し、名前に“SMP001BLogic”およびスーパークラスに“jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic”を入力し、「完了」を押下する。



iii. “SMP001BLogic.java”を以下のように作成する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample001;

import javax.inject.Inject;
import javax.inject.Named;

import jp.terasoluna.batch.tutorial.common.NyusyukkinData;
import jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic;
import jp.terasoluna.fw.batch.blogic.vo.BLogicParam;
import jp.terasoluna.fw.collector.Collector;
import jp.terasoluna.fw.collector.db.DaoCollector;
import jp.terasoluna.fw.collector.util.CollectorUtility;
import jp.terasoluna.fw.file.dao.FileLineWriter;
import jp.terasoluna.fw.file.dao.FileUpdateDAO;

import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Component;

/**
 * ビジネスロジッククラス。(入出金テーブルをcsvファイルに出力するクラス)
 */
@Component
public class SMP001BLogic extends AbstractTransactionBLogic {

    private static final Logger log = LoggerFactory
            .getLogger(SMP001BLogic.class);

    @Inject
    protected SMP001Dao dao;

    @Inject
    @Named("csvFileUpdateDAO")
    protected FileUpdateDAO csvFileUpdateDAO;

    public int doMain(BLogicParam param) {

        // ジョブ終了コード(0:正常終了、-1:異常終了)
        int returnCode = 0;

        // コレクタ
        Collector<NyusyukkinData> collector = new DaoCollector<NyusyukkinData>(
                this.dao, "collectNyusyukkinData", null);

        // ファイル出力用行ライタの取得(入金用)
        FileLineWriter<NyusyukkinData> fileLineWriterNyukin = csvFileUpdateDAO
                .execute("outputFile/SMP001_output_nyukin.csv",
                        NyusyukkinData.class);

        // ファイル出力用行ライタの取得(出金用)
        FileLineWriter<NyusyukkinData> fileLineWriterSyukkin = csvFileUpdateDAO
                .execute("outputFile/SMP001_output_syukkin.csv",
                        NyusyukkinData.class);
```

Inject アノテーションにより DAO のオブジェクトを取得

Named アノテーションにより Inject アノテーションで取得するオブジェクトを指定

DB からデータを取得するコレクタ。
第一引数：SMP001Dao
第二引数：データの取得に利用するメソッド名
第三引数：SQL バインドパラメータ

Writer の取得。
第一引数：ファイルパス
第二引数：ファイル行オブジェクトクラス

```
try {
    // DBから取得したデータを格納するオブジェクト
    NyusyukkinData inputData = null;

    while (collector.hasNext()) {
        // DBからデータを取得
        inputData = collector.next();

        // ファイルへデータを出力(1行)
        // 入出金区分により出力ファイルを変更
        if (inputData != null && inputData.getNyusyukkinKubun() == 0) {
            fileLineWriterNyukin.printDataLine(inputData);
        }
        if (inputData != null && inputData.getNyusyukkinKubun() == 1) {
            fileLineWriterSyukkin.printDataLine(inputData);
        }
    }
} catch (DataAccessException e) {
    if (log.isErrorEnabled()) {
        log.error("データアクセスエラーが発生しました", e);
    }

    returnCode = -1;
} catch (Exception e) {
    if (log.isErrorEnabled()) {
        log.error("エラーが発生しました", e);
    }

    returnCode = -1;
} finally {
    // コレクタのクローズ
    CollectorUtility.closeQuietly(collector);

    // ファイルのクローズ
    CollectorUtility.closeQuietly(fileLineWriterNyukin);
    CollectorUtility.closeQuietly(fileLineWriterSyukkin);
```

```
            // 正常終了時のログ
            if (returnCode == 0 && log.isInfoEnabled()) {
                log.info("ファイル書き込みが正常に終了しました。");
            }
        }

        return returnCode;
    }
}
```

2.2.3. 起動と確認

本節では、作成したジョブの起動と確認方法について説明する。

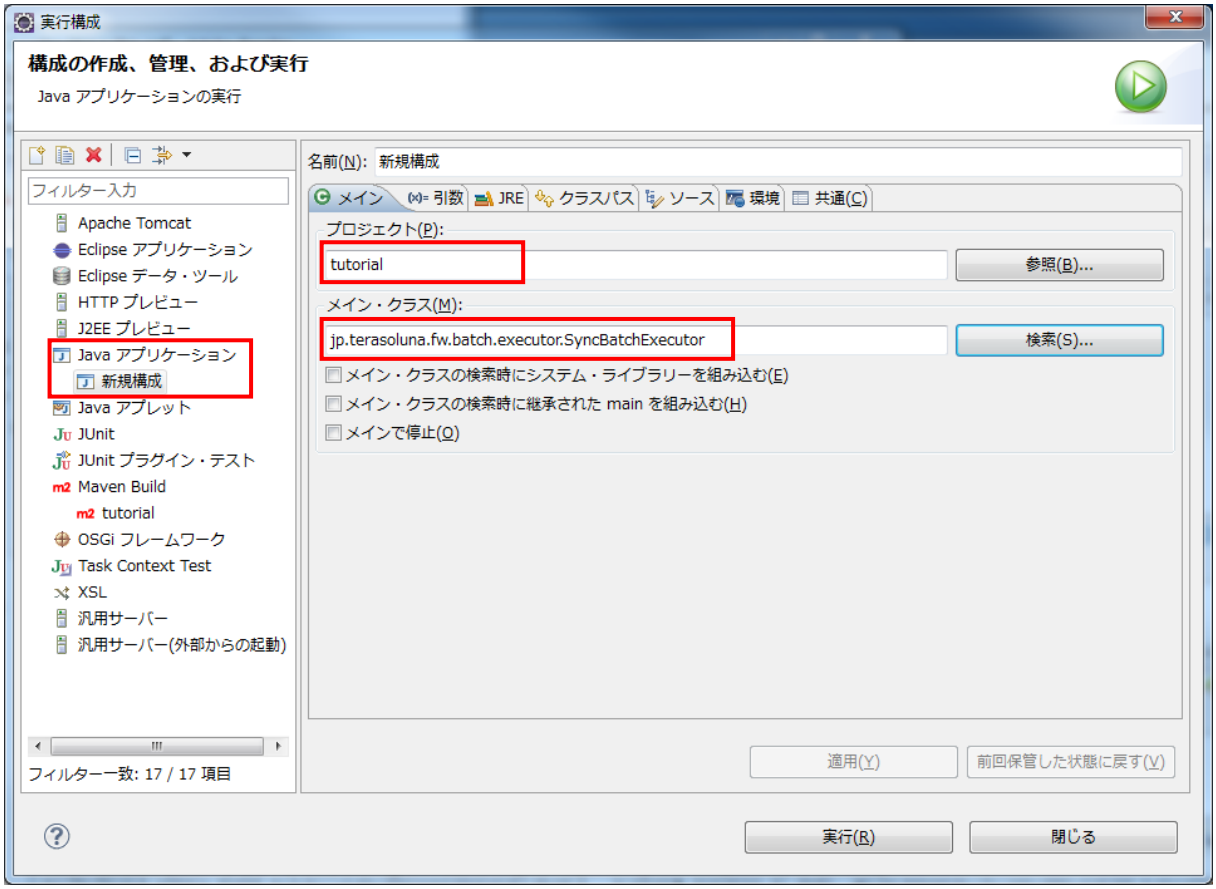
[手順]

(1). ジョブの起動

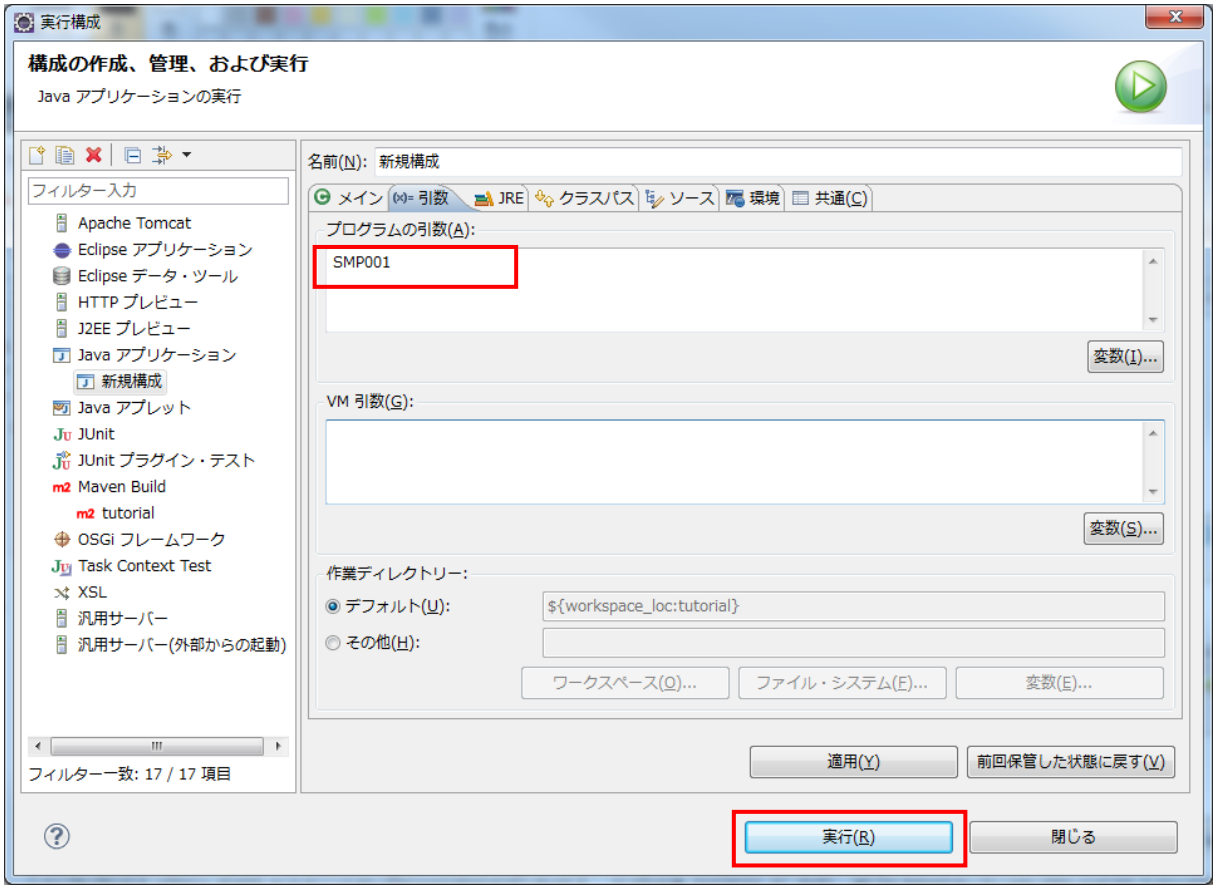
i. メニューより「実行」→「実行構成」を選択する。

ii. 「Java アプリケーション」を右クリックし、「新規」を選択する。

iii. 「Java アプリケーション」直下に追加された「新規構成」を選択し、プロジェクトに“tutorial”、メイン・クラスに“jp.terasoluna.fw.batch.executor.SyncBatchExecutor”を入力する。



iv. 「引数」タブ内の「プログラムの引数」に“SMP001”と入力し、「実行」を押下する。



(2). ログの確認

コンソールに以下のログが出力されることを確認する。

デフォルトの設定では正常終了した場合、ジョブ終了コードは“0”となる。

```
[2015/**/** **:**:**] [main] [j.t.f.b.e.SyncBatchExecutor] [INFO ] [IAL025014]
SyncBatchExecutor START
・・・
[2015/**/** **:**:**] [main] [j.t.b.t.s.SMP001BLogic] [INFO ] ファイル書き込みが
正常に終了しました。
[2015/**/** **:**:**] [main] [j.t.f.b.e.SyncBatchExecutor] [INFO ] [IAL025015]
SyncBatchExecutor END blogicStatus:[0]
```

(3). データの確認

“tutorial ¥outputFile”フォルダーの“SMP001_output_nyukin.csv”および“SMP001_output_syukkin.csv”のデータを確認する。下記のような出力結果が得られる。

SMP001_output_nyukin.csv

東京	468		0	206650	20100819
千葉	778		0	583373	20111221
・・・					
埼玉	431		0	391908	20110315

入出金区分が“0”であることを確認。

SMP001_output_syukkin.csv

千葉	601		1	631103	20110403
千葉	712		1	627111	20111107
・・・					
埼玉	521		1	209756	20100601

入出金区分が“1”であることを確認。

2.3 入力がファイル、出力が DB である場合のジョブ（同期型ジョブ実行）

- 必要な作業

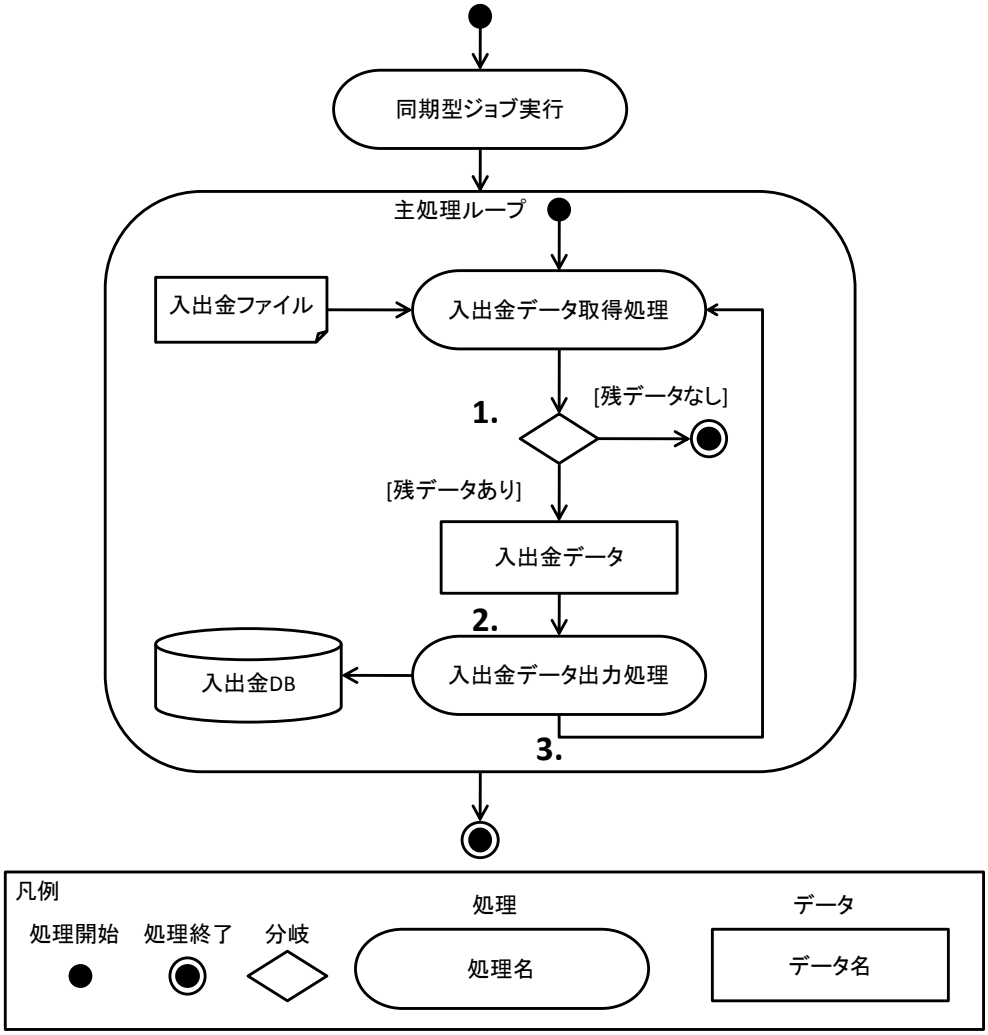
  2.3.1 定義ファイルの作成

  2.3.2 ビジネスロジックの実装

  2.3.3 起動と確認

- ジョブの内容

  図のようにファイルから顧客毎の入出金データを読み込み、入出金データを DB へ出力するジョブを作成する。なお、本節では同期型ジョブ実行を採用する。



1. データを取得し、存在しない場合は処理を終了する。
2. 入出金データを DB へ出力する。
3. 1.へ戻り、次の入出金データを取得する。

- ジョブのリソース

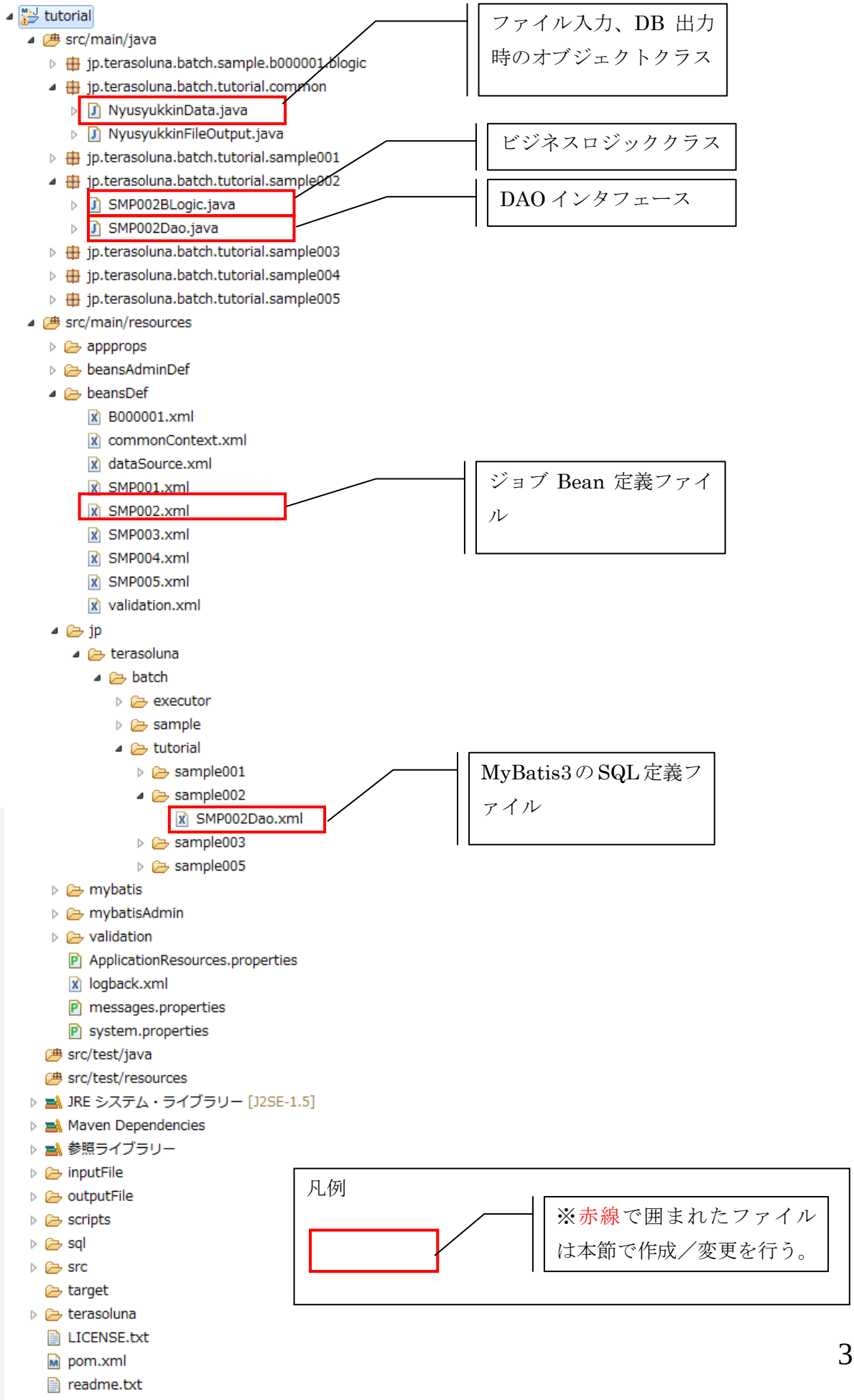
  本ジョブで作成／使用するパッケージ、ジョブ ID は以下の通りである。

  パッケージ：jp.terasoluna.batch.tutorial.common

  jp.terasoluna.batch.tutorial.sample002

  ジョブ ID： SMP002

  また、本ジョブ作成後のプロジェクトは以下のようになる。


tutorial
src/main/java
jp.terasoluna.batch.sample.b000001.blogic
jp.terasoluna.batch.tutorial.common
NyusyukkinData.java
NyusyukkinFileOutput.java
jp.terasoluna.batch.tutorial.sample001
jp.terasoluna.batch.tutorial.sample002
SMP002BLogic.java
SMP002Dao.java
jp.terasoluna.batch.tutorial.sample003
jp.terasoluna.batch.tutorial.sample004
jp.terasoluna.batch.tutorial.sample005
src/main/resources
appprops
beansAdminDef
beansDef
B000001.xml
commonContext.xml
dataSource.xml
SMP001.xml
SMP002.xml
SMP003.xml
SMP004.xml
SMP005.xml
validation.xml
jp
terasoluna
batch
executor
sample
tutorial
sample001
sample002
SMP002Dao.xml
sample003
sample005
mybatis
mybatisAdmin
validation
ApplicationResources.properties
logback.xml
messages.properties
system.properties
src/test/java
src/test/resources
JRE システム・ライブラリー [J2SE-1.5]
Maven Dependencies
参照ライブラリー
inputFile
outputFile
scripts
sql
src
target
terasoluna
LICENSE.txt
pom.xml
readme.txt
ファイル入力、DB 出力時のオブジェクトクラス
ビジネスロジッククラス
DAO インタフェース
ジョブ Bean 定義ファイル
MyBatis3 の SQL 定義ファイル
凡例
※赤線で囲まれたファイルは本節で作成／変更を行う。

2.3.1. 定義ファイルの作成

- ジョブ Bean 定義ファイル
- マッピングファイル

1. ジョブ Bean 定義ファイル

[手順]

(1). “SMP002.xml” を作成

i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources¥beansDef” フォルダーを右クリックする。

ii. 「新規」→「ファイル」を選択し、ファイル名に “SMP002.xml” と入力し「完了」を押下する。

iii. “SMP002.xml” に “tutorial¥src¥main¥resources¥beansDef¥B000001.xml” の内容をコピーし、以下のように変更する。

変更前

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.sample.b000001"/>
```

変更後

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.tutorial.sample002"/>

<!-- SMP002Dao設定 -->
<bean class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"
        value="jp.terasoluna.batch.tutorial.sample002.SMP002Dao" />
    <property name="sqlSessionTemplate" ref="sqlSessionTemplate" />
</bean>
```

2. マッピングファイル

MyBatis3 の定義ファイルであり、SQL 定義情報を記述する。

[手順]

(1). “SMP002Dao.xml” を作成

i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources” フォルダーを右クリックする。

ii. 「新規」→「ファイル」を選択し、「親フォルダーを入力または選択」に “tutorial/src/main/resources/jp/terasoluna/batch/tutorial/sample002”、ファイル名に “SMP002Dao.xml” と入力し「完了」を押下する。

iii. “SMP002Dao.xml” を以下のように編集する。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org/DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="jp.terasoluna.batch.tutorial.sample002.SMP002Dao">
    <!-- データ挿入 -->
    <!-- 入出金テーブル -->
    <insert id="insertNyusyukkinData"
        parameterType="jp.terasoluna.batch.tutorial.common.NyusyukkinData">
        INSERT
        INTO
        NYUSYUKKINTBL (
        SHITENNAME
        ,KOKYAKUID
        ,NYUSYUKKINKUBUN
        ,KINGAKU
        ,TORIHIKIBI
        )
        VALUES (
        #{shitenName}
        ,#{kokyakuId}
        ,#{nyusyukkinKubun}
        ,#{kingaku}
        ,#{torihikibi}
        )
    </insert>
</mapper>
```

2.3.2. ビジネスロジックの実装

本節では、ビジネスロジックの実装方法について説明する。

[手順]

(1). “NyusyukkinData.java” を作成

「2.2.2 ビジネスロジックの実装」の手順(1)を参照のこと。

(2). “SMP002Dao.java” を作成

DAO を作成する。

i. パッケージエクスプローラーで “tutorial” を右クリックする。

ii. 「新規」→「パッケージ」を選択し、名前に “jp.terasoluna.batch.tutorial.sample002” を入力し、「完了」を押下する。

iii. パッケージエクスプローラーで、
“jp.terasoluna.batch.tutorial.sample002”パッケージを右クリックする。
iv. 「新規」→「インターフェース」を選択し、名前に“SMP002Dao”を入力し、「完了」を押下する。
v. “SMP002Dao.java”を以下のように作成する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample002;

import jp.terasoluna.batch.tutorial.common.NyusyukkinData;

public interface SMP002Dao {

    /**
     * 入出金情報を1件挿入する。
     * @param data 入出金情報
     * @return 挿入件数
     */
    public int insertNyusyukkinData(NyusyukkinData data);

}
```

(3). “SMP002BLogic.java”を作成
ビジネスロジックを作成する。
i. パッケージエクスプローラーで、
“jp.terasoluna.batch.tutorial.sample002”パッケージを右クリックする。
ii. 「新規」→「クラス」を選択し、名前に“SMP002BLogic”およびスーパークラスに“jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic”を入力し、「完了」を押下する。
iii. “SMP002BLogic.java”を以下のように作成する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample002;

import javax.inject.Inject;
import javax.inject.Named;
```

```
import jp.terasoluna.batch.tutorial.common.NyusyukkinData;
import jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic;
import jp.terasoluna.fw.batch.blogic.vo.BLogicParam;
import jp.terasoluna.fw.collector.Collector;
import jp.terasoluna.fw.collector.file.FileCollector;
import jp.terasoluna.fw.collector.util.CollectorUtility;
import jp.terasoluna.fw.file.dao.FileQueryDAO;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Component;

/**
 * ビジネスロジッククラス。(CSVファイルを読み込み、DBにデータを挿入する)
 */
@Component
public class SMP002BLogic extends AbstractTransactionBLogic {

    private static final Logger log = LoggerFactory
            .getLogger(SMP002BLogic.class);

    @Inject
    protected SMP002Dao dao;

    @Inject
    @Named("csvFileQueryDAO")
    protected FileQueryDAO csvFileQueryDAO;

    public int doMain(BLogicParam param) {

        // ジョブ終了コード(0:正常終了、-1:異常終了)
        int returnCode = 0;

        // コレクタ
        Collector<NyusyukkinData> collector = new FileCollector<NyusyukkinData>(
                this.csvFileQueryDAO, "inputFile/SMP002_input.csv",
                NyusyukkinData.class);
```

第一引数：FileDAO
第二引数：ファイルパス
第三引数：ファイル行オブジェクトクラス

```
        try {
            // ファイルから取得したデータを格納するオブジェクト
            NyusyukkinData inputData = null;

            while (collector.hasNext()) {
                // ファイルからデータを取得
                inputData = collector.next();

                // DB更新処理
                dao.insertNyusyukkinData(inputData);
            }

        } catch (DataAccessException e) {
            if (log.isErrorEnabled()) {
                log.error("データアクセスエラーが発生しました", e);
            }

            returnCode = -1;
        } catch (Exception e) {
            if (log.isErrorEnabled()) {
                log.error("エラーが発生しました", e);
            }

            returnCode = -1;
        } finally {
            // コレクタのクローズ
            CollectorUtility.closeQuietly(collector);

            // 正常終了時にログ残し
            if (returnCode == 0 && log.isInfoEnabled()) {
                log.info("DBの更新が正常に終了しました。");
            }
        }

        return returnCode;
    }
}
```

2.3.3. 起動と確認

本節では、作成したジョブの起動と確認方法について説明する。

[手順]

(1). ジョブの起動

i. メニューより「実行」→「実行構成」を選択する。

ii. 「Java アプリケーション」を右クリックし、「新規」を選択する。

iii. 「Java アプリケーション」直下に追加された「新規構成」を選択し、プロジェクトに“tutorial”、メイン・クラスに“jp.terasoluna.fw.batch.executor.SyncBatchExecutor”を入力する。

iv. 「引数」タブ内の「プログラムの引数」に“SMP002”と入力し、「実行」を押下する。

(2). ログの確認

コンソールに以下のログが出力されることを確認する。

デフォルトの設定では正常終了した場合、ジョブ終了コードは“0”となる。

```
[2015/**/** **:**:**] [main] [j.t.f.b.e.SyncBatchExecutor] [INFO ] [IAL025014]
SyncBatchExecutor START
・・・
[2015/**/** **:**:**] [main] [j.t.b.t.s.SMP002BLogic] [INFO ] DBの更新が正常に終了しました。
[2015/**/** **:**:**] [main] [j.t.f.b.e.SyncBatchExecutor] [INFO ] [IAL025015]
SyncBatchExecutor END blogicStatus:[0]
```

(3). データの確認

i. Windows のスタートメニューから[プログラム]→[PostgreSQL 9.3]→[pgAdmin III]を起動する。

ii. 「PostgreSQL 9.3(localhost:5432)」を選択し、右クリックメニューから「接続」を選択する。パスワード入力ウィンドウが表示されたら、「P0stgres[ﾋﾟｰｾﾞﾛｴｽﾃｨｰｼﾞｰｱｰﾙｲｰｴｽ]」と入力する。

iii. 画面左のメニューから「データベース(2)」「terasoluna」を選択し、「スキーマ(1)」→「public」→「テーブル(2)」と選択する。

iv. 「nyusyukkintbl」で右クリックし、「データビュー」→「全ての列を表示」をクリックし、テーブルのデータを確認する。下記のような結果が得られる。

	shitenname character va	kokyakuid character va	nyusyukkink character va	kingaku bigint	torihikibi date
1	千葉	0624	1	337157	2010-11-17
2	東京	0468	0	206650	2010-08-19
3	埼玉	0493	1	190376	2010-10-05
			⋮		
150	埼玉	521	1	209756	2010-06-01

2.4 入力チェック機能を利用した、入力がファイル、出力が DB である場合のジョブ(同期型ジョブ実行)

- 必要な作業

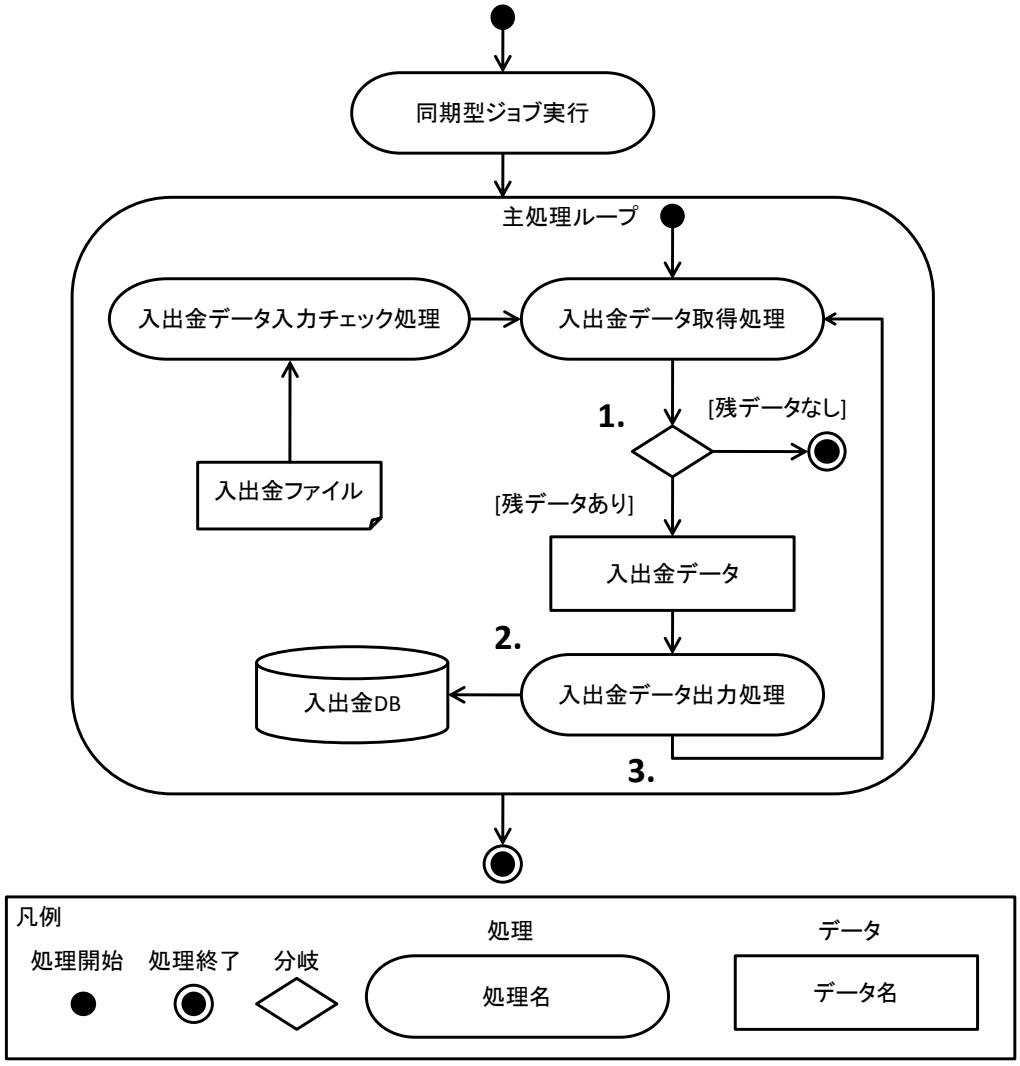
  2.4.1 定義ファイルの作成

  2.4.2 ビジネスロジックの実装

  2.4.3 起動と確認

- ジョブの内容

  図のようにファイルから顧客毎の入出金データを読み込み、入力チェックを行い DB へ出力するジョブを作成する。なお、本節では同期型ジョブ実行を採用する。



1. フレームワークが入力チェックした後のデータを取得し、存在しない場合は処理を終了する。
2. 入出金データを DB へ出力する。
3. 1.へ戻り、次の入出金データを取得する。

- ジョブのリソース

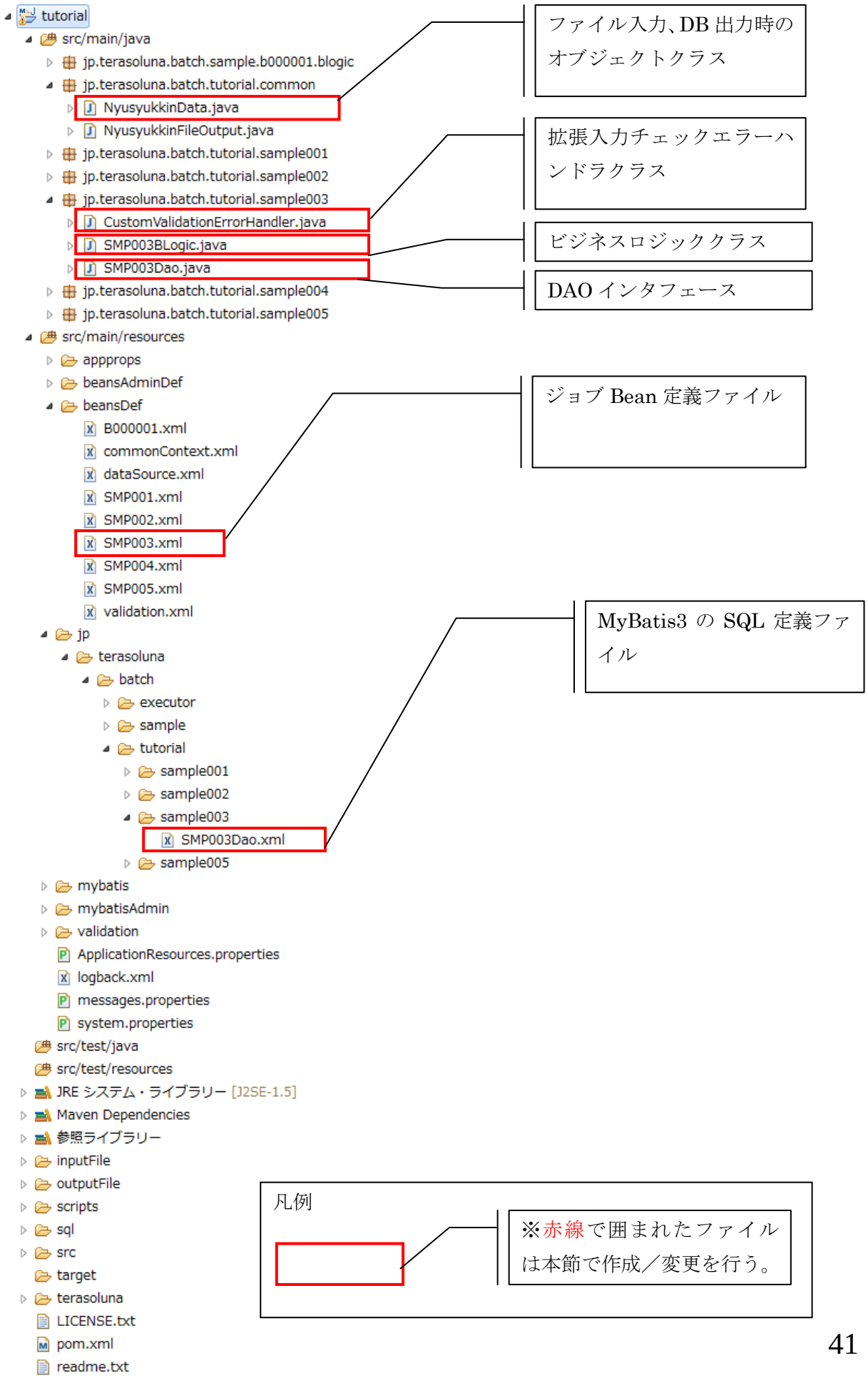
  本ジョブで作成／使用するパッケージ、ジョブ ID は以下の通りである。

  パッケージ：jp.terasoluna.batch.tutorial.common

  jp.terasoluna.batch.tutorial.sample003

  ジョブ ID： SMP003

  また、本ジョブ作成後のプロジェクトは以下のようになる。


tutorial
src/main/java
jp.terasoluna.batch.sample.b000001.blogic
jp.terasoluna.batch.tutorial.common
NyusyukkinData.java
NyusyukkinFileOutput.java
jp.terasoluna.batch.tutorial.sample001
jp.terasoluna.batch.tutorial.sample002
jp.terasoluna.batch.tutorial.sample003
CustomValidationErrorHandler.java
SMP003BLogic.java
SMP003Dao.java
jp.terasoluna.batch.tutorial.sample004
jp.terasoluna.batch.tutorial.sample005
src/main/resources
appprops
beansAdminDef
beansDef
B000001.xml
commonContext.xml
dataSource.xml
SMP001.xml
SMP002.xml
SMP003.xml
SMP004.xml
SMP005.xml
validation.xml
jp
terasoluna
batch
executor
sample
tutorial
sample001
sample002
sample003
SMP003Dao.xml
sample005
mybatis
mybatisAdmin
validation
ApplicationResources.properties
logback.xml
messages.properties
system.properties
src/test/java
src/test/resources
JRE システム・ライブラリー [J2SE-1.5]
Maven Dependencies
参照ライブラリー
inputFile
outputFile
scripts
sql
src
target
terasoluna
LICENSE.txt
pom.xml
readme.txt
ファイル入力、DB 出力時のオブジェクトクラス
拡張入力チェックエラーハンドラクラス
ビジネスロジッククラス
DAO インタフェース
ジョブ Bean 定義ファイル
MyBatis3 の SQL 定義ファイル
凡例
※赤線で囲まれたファイルは本節で作成／変更を行う。

2.4.1. 定義ファイルの作成

- ジョブ Bean 定義ファイル
- マッピングファイル

1. ジョブ Bean 定義ファイル

[手順]

(1). “SMP003.xml” を作成

i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources¥beansDef” フォルダーを右クリックする。

ii. 「新規」→「ファイル」を選択し、ファイル名に “SMP003.xml” と入力し「完了」を押下する。

iii. “SMP003.xml” に “tutorial¥src¥main¥resources¥beansDef¥B000001.xml” の内容をコピーし、以下のように変更する。

変更前

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.sample.b000001"/>
```

変更後

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.tutorial.sample003"/>

<!-- SMP003Dao設定 -->
<bean class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"
value="jp.terasoluna.batch.tutorial.sample003.SMP003Dao" />
    <property name="sqlSessionTemplate" ref="sqlSessionTemplate" />
</bean>
```

2. マッピングファイル

MyBatis3 の定義ファイルであり、SQL 定義情報を記述する。

[手順]

(2). “SMP002Dao.xml” を作成

i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources” フォルダーを右クリックする。

ii. 「新規」→「ファイル」を選択し、「親フォルダーを入力または選択」に “tutorial/src/main/resources/jp/terasoluna/batch/tutorial/sample003”、ファイル名に “SMP003Dao.xml” と入力し「完了」を押下する。

iii. “SMP003Dao.xml” を以下のように編集する。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org/DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="jp.terasoluna.batch.tutorial.sample003.SMP003Dao">
    <!-- データ挿入 -->
    <!-- 入出金テーブル -->
    <insert id="insertNyusyukkinData"
        parameterType="jp.terasoluna.batch.tutorial.common.NyusyukkinData">
        INSERT
        INTO
        NYUSYUKKINTBL (
        SHITENNAME
        ,KOKYAKUID
        ,NYUSYUKKINKUBUN
        ,KINGAKU
        ,TORIHIKIBI
        )
        VALUES (
        #{shitenName}
        ,#{kokyakuId}
        ,#{nyusyukkinKubun}
        ,#{kingaku}
        ,#{torihikibi}
        )
    </insert>
</mapper>
```

2.4.2. ビジネスロジックの実装

本節では、ビジネスロジックの実装方法について説明する。

[手順]

(1). “NyusyukkinData.java”を作成

i. 「2.2.2 ビジネスロジックの実装」の手順(1)を参照し、“NyusyukkinData.java”を作成する。

ii. BeanValidation を利用した入力チェックを実行するため、作成した“NyusyukkinData.java”に対して以下のように入力チェックルールを追加する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.common;

import java.util.Date;

import javax.validation.constraints.NotNull;

import jp.terasoluna.fw.file.annotation.FileFormat;
import jp.terasoluna.fw.file.annotation.InputFileColumn;
import jp.terasoluna.fw.file.annotation.OutputFileColumn;

/**
 * 入出金情報のパラメータクラス。
 */
@FileFormat(overWriteFlg = true, fileEncoding = "MS932")
public class NyusyukkinData {

    /**
     * 支店名
     */
    @InputFileColumn(columnIndex = 0)
    @OutputFileColumn(columnIndex = 0)
    private String shitenName;

    /**
     * 顧客ID
     */
    @InputFileColumn(columnIndex = 1)
    @OutputFileColumn(columnIndex = 1)
    @NotEmpty
    private String kokyakuId;

    /**
     * 入出金区分 0:出金 1:入金
     */
```

この例では、顧客 ID を必須項目とする。

```
    @InputFileColumn(columnIndex = 2)
    @OutputFileColumn(columnIndex = 2)
    private int nyusyukkinKubun;

    /**
     * 取引金額
     */
    @InputFileColumn(columnIndex = 3)
    @OutputFileColumn(columnIndex = 3)
    private int kingaku;

    /**
     * 取引日
     */
    @InputFileColumn(columnIndex = 4, columnFormat = "yyyyMMdd")
    @OutputFileColumn(columnIndex = 4, columnFormat = "yyyyMMdd")
    private Date torihikibi;

    /**
     * 支店名を取得する。
     * @return shitenName
     */
    public String getShitenName() {
        return shitenName;
    }

    /**
     * 支店名を設定する。
     * @param shitenName
     */
    public void setShitenName(String shitenName) {
        this.shitenName = shitenName;
    }

    /**
     * 顧客IDを取得する。
     * @return kokyakuId
     */
```

```
    public String getKokyakuId() {
        return kokyakuId;
    }

    /**
     * 顧客IDを設定する。
     * @param kokyakuId
     */
    public void setKokyakuId(String kokyakuId) {
        this.kokyakuId = kokyakuId;
    }

    /**
     * 入出金区分を取得する。
     * @return nyusyukkinKubun
     */
    public int getNyusyukkinKubun() {
        return nyusyukkinKubun;
    }

    /**
     * 入出金区分を設定する。
     * @param nyusyukkinKubun
     */
    public void setNyusyukkinKubun(int nyusyukkinKubun) {
        this.nyusyukkinKubun = nyusyukkinKubun;
    }

    /**
     * 取引金額を取得する。
     * @return kingaku
     */
    public int getKingaku() {
        return kingaku;
    }

    /**
     * 取引金額を設定する。
```

```
     * @param kingaku
     */
    public void setKingaku(int kingaku) {
        this.kingaku = kingaku;
    }

    /**
     * 取引日を取得する。
     * @return torihikibi
     */
    public Date getTorihikibi() {
        return torihikibi;
    }

    /**
     * 取引日を設定する。
     * @param torihikibi
     */
    public void setTorihikibi(Date torihikibi) {
        this.torihikibi = torihikibi;
    }
}
```

(2). “CustomValidationErrorHandler.java”を作成

拡張入力チェックエラーハンドラクラスを作成する。

i. パッケージエクスプローラーで“tutorial”を右クリックする。

ii. 「新規」→「パッケージ」を選択し、名前に“jp.terasoluna.batch.tutorial.sample003”を入力し、「完了」を押下する。

iii. パッケージエクスプローラーで、“jp.terasoluna.batch.tutorial.sample003”パッケージを右クリックする。

iv. 「新規」→「クラス」を選択し、名前に“CustomValidationErrorHandler”およびインターフェースに“jp.terasoluna.fw.collector.validate.ValidationErrorHandler”を入力し、「完了」を押下する。

v. “CustomValidationErrorHandler.java”を以下のように作成する。

```java
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample003;

import java.util.List;

import jp.terasoluna.fw.collector.validate.ValidateErrorStatus;
import jp.terasoluna.fw.collector.validate.ValidationErrorHandler;
import jp.terasoluna.fw.collector.vo.DataValueObject;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.validation.Errors;
import org.springframework.validation.FieldError;

public class CustomValidationErrorHandler implements ValidationErrorHandler {

    /**
     * Log.
     */
    private static final Logger log = LoggerFactory
            .getLogger(CustomValidationErrorHandler.class);

    public ValidateErrorStatus handleValidationError(
            DataValueObject dataValueObject, Errors errors) {

        if (log.isWarnEnabled()) {
            List<FieldError> fieldErrorList = errors.getFieldErrors();
            for (FieldError fieldError : fieldErrorList) {
                log.warn("{} フィールドにおいて必須入力チェックエラー発生",
                        fieldError.getField());
            }
        }

        // エラーデータをとばして、そのまま処理を続行する
        return ValidateErrorStatus.SKIP;
```

他に
END：処理を停止する
CONTINUE:エラーデータをとばさず、そのまま処理を続行する

```
    }
}
```

(3). “SMP003Dao.java”を作成

DAO を作成する。

i. パッケージエクスプローラーで、
“jp.terasoluna.batch.tutorial.sample003”パッケージを右クリックする。

ii. 「新規」→「インターフェース」を選択し、名前に“SMP003Dao”を入力し、「完了」を押下する。

iii. “SMP003Dao.java”を以下のように作成する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample003;

import jp.terasoluna.batch.tutorial.common.NyusyukkinData;

public interface SMP003Dao {

    /**
     * 入出金情報を1件挿入する。
     * @param data 入出金情報
     * @return 挿入件数
     */
    public int insertNyusyukkinData(NyusyukkinData data);

}
```

(4). “SMP003BLogic.java”を作成

ビジネスロジックを作成する。

i. パッケージエクスプローラーで、
“jp.terasoluna.batch.tutorial.sample003”パッケージを右クリックする。

ii. 「新規」→「クラス」を選択し、名前に“SMP003BLogic”およびスーパークラスに“jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic”を入力し、「完了」を押下する。

iii. “SMP003BLogic.java”を以下のように作成する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
```

```
package jp.terasoluna.batch.tutorial.sample003;

import javax.inject.Inject;
import javax.inject.Named;

import jp.terasoluna.batch.tutorial.common.NyusyukkinData;
import jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic;
import jp.terasoluna.fw.batch.blogic.vo.BLogicParam;
import jp.terasoluna.fw.collector.Collector;
import jp.terasoluna.fw.collector.file.FileValidateCollector;
import jp.terasoluna.fw.collector.util.CollectorUtility;
import jp.terasoluna.fw.file.dao.FileQueryDAO;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Component;
import org.springframework.validation.Validator;


/**
 * ビジネスロジッククラス。(CSVファイルを読み込み、DBにデータを挿入する)
 */
@Component
public class SMP003BLogic extends AbstractTransactionBLogic {

    private static final Logger log = LoggerFactory.getLogger(SMP003BLogic.class);

    @Inject
    protected SMP003Dao dao;

    @Inject
    @Named("csvFileQueryDAO")
    protected FileQueryDAO csvFileQueryDAO;

    @Inject
    protected Validator validator;
```

Inject アノテーションにより入力チェック Validator を取得

```
public int doMain(BLogicParam param) {

    // ジョブ終了コード(0:正常終了、-1:異常終了)
    int returnCode = 0;

    CustomValidationErrorHandler customValidationErrorHandler
        = new CustomValidationErrorHandler();

    // コレクタ
    Collector<NyusyukkinData> collector = new FileValidateCollector<NyusyukkinData>(
            this.csvFileQueryDAO, "inputFile/SMP003_input.csv",
            NyusyukkinData.class, validator, customValidationErrorHandler);

    try {
        // ファイルから取得したデータを格納するオブジェクト
        NyusyukkinData inputData = null;
        while (collector.hasNext()) {

            // ファイルからデータを取得
            inputData = collector.next();

            // DB更新処理
            dao.insertNyusyukkinData(inputData);
        }
    } catch (DataAccessException e) {
        if (log.isErrorEnabled()) {
            log.error("データアクセスエラーが発生しました", e);
        }

        returnCode = -1;
    } catch (Exception e) {
        if (log.isErrorEnabled()) {
            log.error("エラーが発生しました", e);
        }

        returnCode = -1;
    } finally {
        // コレクタのクローズ
```

拡張入力チェックエラーハンドラクラスのインスタンスを生成

第四引数：Validator
第五引数:拡張入力チェックエラーハンドラクラス

```
            CollectorUtility.closeQuietly(collector);

            // 正常終了時にログ残し
            if (returnCode == 0 && log.isInfoEnabled()) {
                log.info("DBの更新が正常に終了しました。");
            }
        }

        return returnCode;
    }
}
```

2.4.3.起動と確認

本節では、作成したジョブの起動と確認方法について説明する。

[手順]

(1). ジョブの起動

i. メニューより「実行」→「実行構成」を選択する。

ii. 「Java アプリケーション」を右クリックし、「新規」を選択する。

iii. 「Java アプリケーション」直下に追加された「新規構成」を選択し、プロジェクトに“tutorial”、メイン・クラスに“jp.terasoluna.fw.batch.executor.SyncBatchExecutor”を入力する。

iv. 「引数」タブ内の「プログラムの引数」に“SMP003”を入力し、「実行」を押下する。

(2). ログの確認

コンソールに以下のログが出力されることを確認する。

デフォルトの設定では正常終了した場合、ジョブ終了コードは“0”となる。

```
[2015/**/** **:**:**] [main] [j.t.f.b.e.SyncBatchExecutor] [INFO ] [IAL025014]
SyncBatchExecutor START
・・・
[2015/**/** **:**:**] [CollectorThreadFactory-1-thread-1]
[j.t.b.t.s.CustomValidationErrorHandler] [WARN ] kokyakuIdフィールドにおいて必須
入力チェックエラー発生
・・・
[2015/**/** **:**:**] [main] [j.t.b.t.s.SMP003BLogic] [INFO ] DBの更新が正常に終
了しました。
[2015/**/** **:**:**] [main] [j.t.f.b.e.SyncBatchExecutor] [INFO ] [IAL025015]
SyncBatchExecutor END blogicStatus:[0]
```

2.5 コントロールブレイク機能を利用した、入力がファイル、出力がファイルである場合のジョブ(同期型ジョブ実行)

- 必要な作業

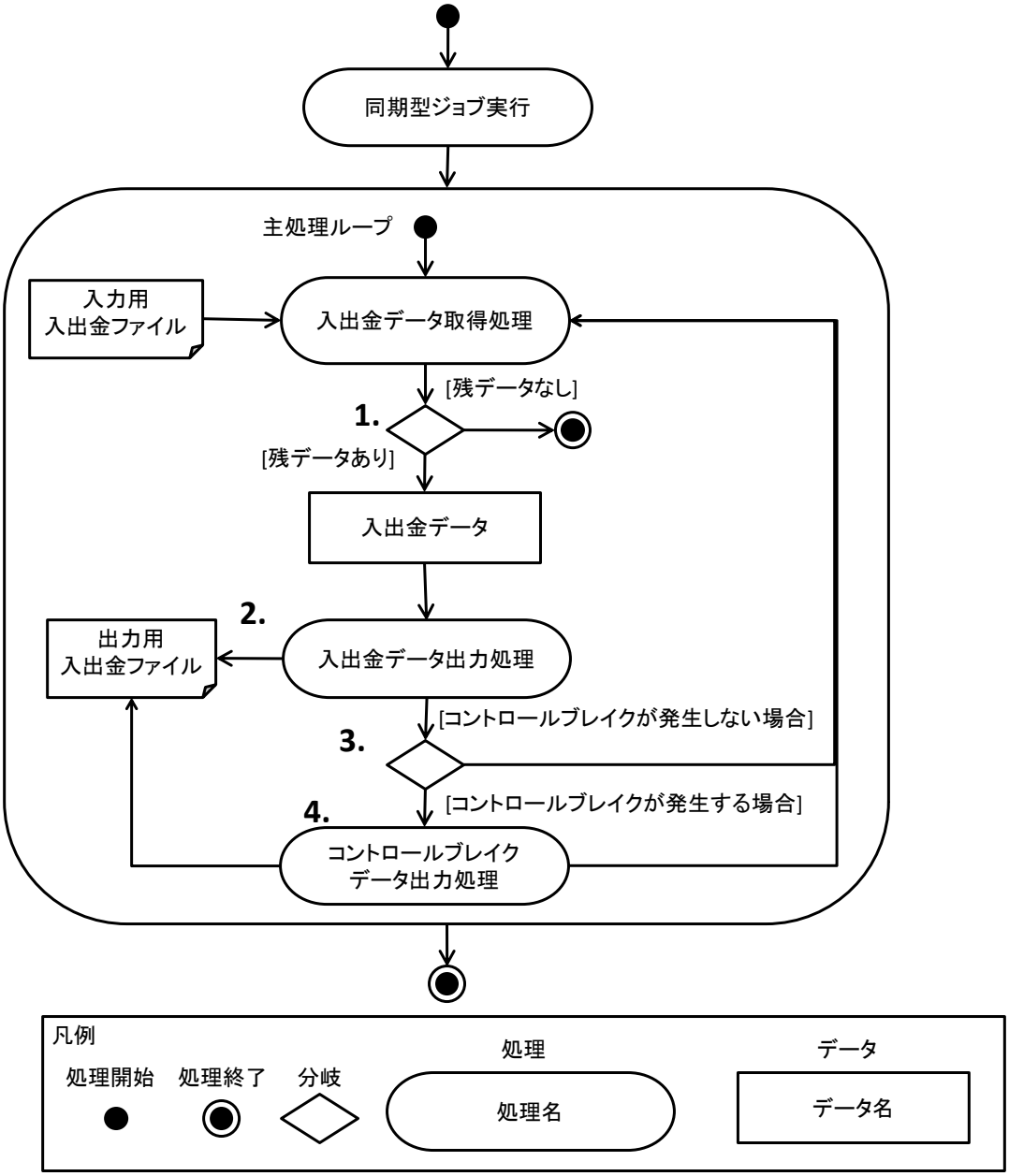
  2.5.1 定義ファイルの作成

  2.5.2 ビジネスロジックの実装

  2.5.3 起動と確認

- ジョブの内容

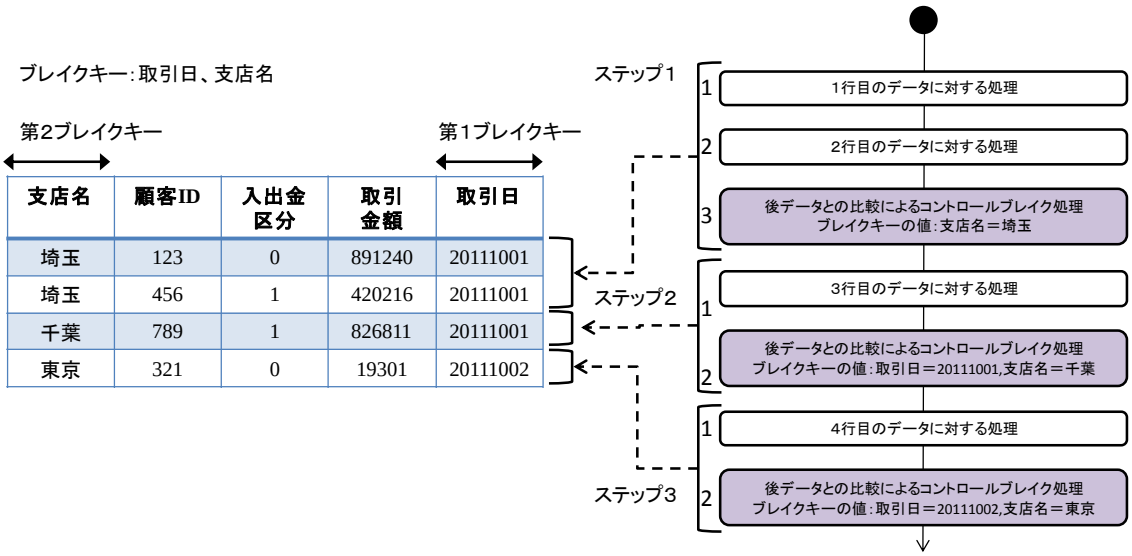
  図のようにファイルから顧客毎の入出金データを読み込み、ファイルへ出力するジョブを作成する。コントロールブレイク処理とは、ある項目をキーとして、キーが変わるまで集計したり見出しを追加したりする処理である。なお、本節では同期型ジョブ実行を採用する



1. 入出金データを取得し、存在しない場合は処理を終了する。
2. 入出金データをファイルへ出力する。
3. コントロールブレイクが発生しない場合は１．へ戻り、コントロールブレイクが発生する場合はコントロールブレイク時のデータをファイルへ出力する。
4. 1. へ戻り、次の入出金データを取得する。

- コントロールブレイク処理の概要

  コントロールブレイク時に行われる処理の概要については下記の通りである。



ステップ		処理内容
1	1	1 行目に対する処理が実行される。
	2	2 行目に対する処理が実行される。
	3	後データとの比較によるコントロールブレイク処理が実行される
2	1	3 行目に対する処理が実行される。
	2	後データとの比較によるコントロールブレイク処理が実行される
3	1	4 行目に対する処理が実行される。
	2	後データとの比較によるコントロールブレイク処理が実行される

  - 設定したブレイクキーの値が切り替わったタイミングでコントロールブレイク処理が実行される。
  - ビジネスロジック中でブレイクキーを取得することにより、ビジネスロジック中で切り替わった値を取得することが可能である。

- ジョブのリソース

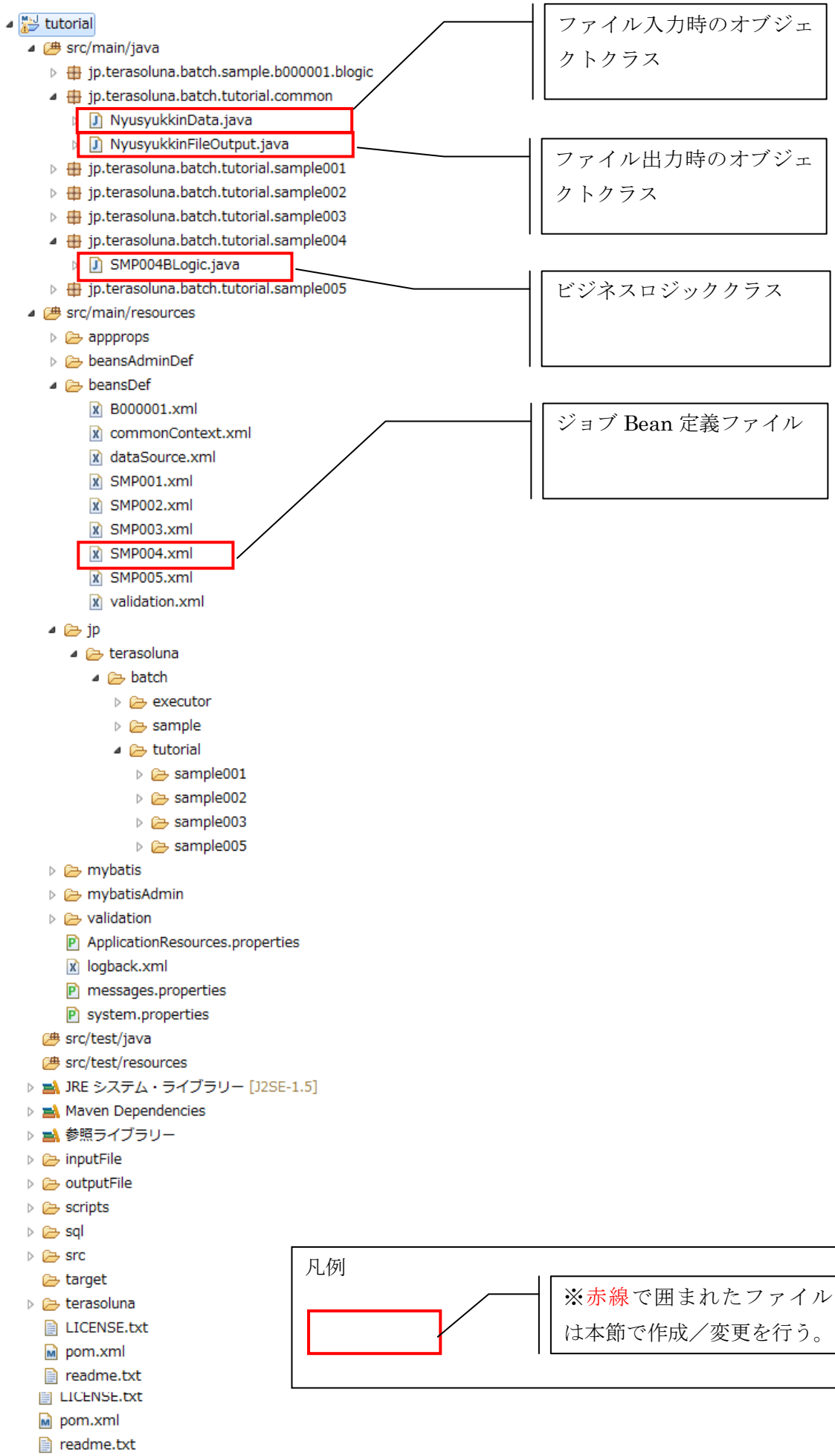
  本ジョブで作成／使用するパッケージ、ジョブ ID は以下の通りである。

  パッケージ：jp.terasoluna.batch.tutorial.common

  jp.terasoluna.batch.tutorial.sample004

  ジョブ ID： SMP004

  また、本ジョブ作成後のプロジェクトは以下のようになる。


tutorial
src/main/java
jp.terasoluna.batch.sample.b000001.blogic
jp.terasoluna.batch.tutorial.common
NyusyukkinData.java
NyusyukkinFileOutput.java
jp.terasoluna.batch.tutorial.sample001
jp.terasoluna.batch.tutorial.sample002
jp.terasoluna.batch.tutorial.sample003
jp.terasoluna.batch.tutorial.sample004
SMP004BLogic.java
jp.terasoluna.batch.tutorial.sample005
src/main/resources
appprops
beansAdminDef
beansDef
B000001.xml
commonContext.xml
dataSource.xml
SMP001.xml
SMP002.xml
SMP003.xml
SMP004.xml
SMP005.xml
validation.xml
jp
terasoluna
batch
executor
sample
tutorial
sample001
sample002
sample003
sample005
mybatis
mybatisAdmin
validation
ApplicationResources.properties
logback.xml
messages.properties
system.properties
src/test/java
src/test/resources
JRE システム・ライブラリー [J2SE-1.5]
Maven Dependencies
参照ライブラリー
inputFile
outputFile
scripts
sql
src
target
terasoluna
LICENSE.txt
pom.xml
readme.txt
LICENSE.txt
pom.xml
readme.txt
ファイル入力時のオブジェクトクラス
ファイル出力時のオブジェクトクラス
ビジネスロジッククラス
ジョブ Bean 定義ファイル
凡例
※赤線で囲まれたファイルは本節で作成／変更を行う。

2.5.1. 定義ファイルの作成

- ジョブ Bean 定義ファイル

1. ジョブ Bean 定義ファイル

[手順]

(1). “SMP004.xml” を作成

i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources¥beansDef” フォルダーを右クリックする。

ii. 「新規」→「ファイル」を選択し、ファイル名に “SMP004.xml” と入力し「完了」を押下する。

iii. “SMP004.xml” に “tutorial¥src¥main¥resources¥beansDef¥B000001.xml” の内容をコピーし、以下のように変更する。

変更前

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.sample.b000001"/>
```

変更後

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.tutorial.sample004"/>
```

2.5.2. ビジネスロジックの実装

本節では、ビジネスロジックの実装方法について説明する。

[手順]

(1). “NyusyukkinData.java” を作成

「2.2.2 ビジネスロジックの実装」の手順(1)を参照のこと。

(2). “NyusyukkinFileOutput.java” を作成

コントロールブレイク時のデータを格納するクラスを作成する。なお、このクラスはファイル出力時のファイル行オブジェクトクラスとなる。そのため、ファイルの個々の項目の定義情報をアノテーションにより設定している。

i. パッケージエクスプローラーで、“jp.terasoluna.batch.tutorial.common” パッケージを右クリックする。

ii. 「新規」→「クラス」を選択し、ファイル名に “NyusyukkinFileOutput” と入力し「完了」を押下する。

iii. “NyusyukkinFileOutput.java” を以下のように編集する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.common;
```

```
import java.util.Date;

import jp.terasoluna.fw.file.annotation.FileFormat;
import jp.terasoluna.fw.file.annotation.OutputFileColumn;

/**
 * 入出金情報のファイル出力パラメータクラス。
 */
@FileFormat(overWriteFlg = true, fileEncoding = "MS932")
public class NyusyukkinFileOutput {

    /**
     * 取引日
     */
    @OutputFileColumn(columnIndex = 0, columnFormat = "yyyyMMdd")
    private Date torihikibi;

    /**
     * 支店名
     */
    @OutputFileColumn(columnIndex = 1)
    private String shitenName;

    /**
     * 入金回数
     */
    @OutputFileColumn(columnIndex = 2)
    private int nyukinNum;

    /**
     * 出金回数
     */
    @OutputFileColumn(columnIndex = 3)
    private int syukkinNum;

    /**
     * 入金合計
     */
```

```
    @OutputFileColumn(columnIndex = 4)
    private int nyukinSum;

    /**
     * 出金合計
     */
    @OutputFileColumn(columnIndex = 5)
    private int syukkinSum;

    /**
     * 取引日を取得する。
     *
     * @return torihikibi
     */
    public Date getTorihikibi() {
        return torihikibi;
    }

    /**
     * 取引日を設定する。
     *
     * @param torihikibi
     */
    public void setTorihikibi(Date torihikibi) {
        this.torihikibi = torihikibi;
    }

    /**
     * 支店名を取得する。
     *
     * @return shitenName
     */
    public String getShitenName() {
        return shitenName;
    }

    /**
     * 支店名を設定する。
```

```
     *
     * @param shitenName
     */
    public void setShitenName(String shitenName) {
        this.shitenName = shitenName;
    }

    /**
     * 入金回数を取得する。
     *
     * @return nyukinNum
     */
    public int getNyukinNum() {
        return nyukinNum;
    }

    /**
     * 入金回数を設定する。
     *
     * @param nyukinNum
     */
    public void setNyukinNum(int nyukinNum) {
        this.nyukinNum = nyukinNum;
    }

    /**
     * 出金回数を取得する。
     *
     * @return syukkinNum
     */
    public int getSyukkinNum() {
        return syukkinNum;
    }

    /**
     * 出金回数を設定する。
     *
     * @param syukkinNum
```

```
     */
    public void setSyukkinNum(int syukkinNum) {
        this.syukkinNum = syukkinNum;
    }

    /**
     * 入金合計を取得する。
     *
     * @return nyukinSum
     */
    public int getNyukinSum() {
        return nyukinSum;
    }

    /**
     * 入金合計を設定する。
     *
     * @param nyukinSum
     */
    public void setNyukinSum(int nyukinSum) {
        this.nyukinSum = nyukinSum;
    }

    /**
     * 出金合計を取得する。
     *
     * @return syukkinSum
     */
    public int getSyukkinSum() {
        return syukkinSum;
    }

    /**
     * 出金合計を設定する。
     *
     * @param syukkinSum
     */
    public void setSyukkinSum(int syukkinSum) {
```

```
        this.syukkinSum = syukkinSum;
    }

}
```

(3). “SMP004BLogic.java”を作成

ビジネスロジックを作成する。

i. パッケージエクスプローラーで“tutorial”を右クリックする。

ii. 「新規」→「パッケージ」を選択し、名前に“jp.terasoluna.batch.tutorial.sample004”を入力し、「完了」を押下する。

iii. パッケージエクスプローラーで、“jp.terasoluna.batch.tutorial.sample004”パッケージを右クリックする。

iv. 「新規」→「クラス」を選択し、名前に“SMP004BLogic”およびスーパークラスに“jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic”を入力し、「完了」を押下する。

v. “SMP004BLogic.java”を以下のように作成する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample004;

import java.util.Date;
import java.util.Map;

import javax.inject.Inject;
import javax.inject.Named;

import jp.terasoluna.batch.tutorial.common.NyusyukkinData;
import jp.terasoluna.batch.tutorial.common.NyusyukkinFileOutput;
import jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic;
import jp.terasoluna.fw.batch.blogic.vo.BLogicParam;
import jp.terasoluna.fw.collector.Collector;
import jp.terasoluna.fw.collector.file.FileCollector;
import jp.terasoluna.fw.collector.util.CollectorUtility;
import jp.terasoluna.fw.collector.util.ControlBreakChecker;
import jp.terasoluna.fw.file.dao.FileLineWriter;
import jp.terasoluna.fw.file.dao.FileQueryDAO;
import jp.terasoluna.fw.file.dao.FileUpdateDAO;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Component;

/**
 * ビジネスロジッククラス。(CSVファイルを読み込み、csvファイルに出力するクラス)
 */
@Component
public class SMP004BLogic extends AbstractTransactionBLogic {

    private static final Logger log = LoggerFactory.getLogger(SMP004BLogic.class);

    @Inject
    @Named("csvFileQueryDAO")
    protected FileQueryDAO csvFileQueryDAO;

    @Inject
    @Named("csvFileUpdateDAO")
    protected FileUpdateDAO csvFileUpdateDAO;

    public int doMain(BLogicParam param) {

        // ジョブ終了コード(0:正常終了、-1:異常終了)
        int returnCode = 0;

        // コレクタ
        Collector<NyusyukkinData> collector = new FileCollector<NyusyukkinData>(
                this.csvFileQueryDAO, "inputFile/SMP004_input.csv",
                NyusyukkinData.class);

        // ファイル出力用行ライタの取得
        FileLineWriter<NyusyukkinFileOutput> fileLineWriter = csvFileUpdateDAO
                .execute("outputFile/SMP004_output.csv",
                        NyusyukkinFileOutput.class);

        try {
```

```
        // ファイルから取得したデータを格納するオブジェクト
        NyusyukkinData inputData = null;

        // 入金のカウント用
        int nyukinNum = 0;
        // 出金のカウント用
        int syukkinNum = 0;
        // 入金合計用
        int nyukinSum = 0;
        // 出金合計用
        int syukkinSum = 0;

        while (collector.hasNext()) {
            // ファイルからデータを取得
            inputData = collector.next();

            // コントロールブレイク判定
            // 支店名、取引日に変更がある場合
            boolean ctrlBreak = ControlBreakChecker.isBreak(collector,
                    "torihikibi", "shitenName");

            // 入出金区分のカウント、合計計算
            if (inputData != null && inputData.getNyusyukkinKubun() == 0) {
                syukkinNum++;
                syukkinSum += inputData.getKingaku();
            } else if (inputData != null
                    && inputData.getNyusyukkinKubun() == 1) {
                nyukinNum++;
                nyukinSum += inputData.getKingaku();
            }

            // コントロールブレイク処理
            if (ctrlBreak) {
                // キーデータをマップで取得
                Map<String, Object> brkMap = ControlBreakChecker
                        .getBreakKey(collector, "torihikibi", "shitenName");
                Date torihikibi = null;
                String shitenName = null;
```

コントロールブレイクの判定を取得
第一引数：コレクタ
第二引数以降：コントロールブレイクキー

コントロールブレイクキーに対する切り替わった値を Map 型で取得する。
第一引数：コレクタ
第二引数以降：コントロールブレイクキー

```
                    if (brkMap.containsKey("torihikibi")) {
                        torihikibi = (Date) brkMap.get("torihikibi");
                    } else {
                        torihikibi = inputData.getTorihikibi();
                    }
                    if (brkMap.containsKey("shitenName")) {
                        shitenName = (String) brkMap.get("shitenName");
                    } else {
                        shitenName = inputData.getShitenName();
                    }

                    // コントロールブレイク時のデータの作成
                    NyusyukkinFileOutput outputData = new NyusyukkinFileOutput();
                    outputData.setTorihikibi(torihikibi);
                    outputData.setShitenName(shitenName);
                    outputData.setNyukinNum(nyukinNum);
                    outputData.setNyukinSum(nyukinSum);
                    outputData.setSyukkinNum(syukkinNum);
                    outputData.setSyukkinSum(syukkinSum);

                    // データをファイルへ出力(1行)
                    fileLineWriter.printDataLine(outputData);

                    // 入出金区分カウントの初期化
                    nyukinNum = 0;
                    syukkinNum = 0;
                    nyukinSum = 0;
                    syukkinSum = 0;
                }
            }
        } catch (DataAccessException e) {
            if (log.isErrorEnabled()) {
                log.error("データアクセスエラーが発生しました", e);
            }

            returnCode = -1;
        } catch (Exception e) {
            if (log.isErrorEnabled()) {
```

```
                log.error("エラーが発生しました", e);
            }

            returnCode = -1;
        } finally {
            // コレクタのクローズ
            CollectorUtility.closeQuietly(collector);

            // ファイルのクローズ
            CollectorUtility.closeQuietly(fileLineWriter);

            // 正常終了時のログ
            if (returnCode == 0 && log.isInfoEnabled()) {
                log.info("ファイル書き込みが正常に終了しました。");
            }
        }

        return returnCode;
    }
}
```

2.5.3. 起動と確認

本節では、作成したジョブの起動と確認方法について説明する。

[手順]

(1). ジョブの起動

i. メニューより「実行」→「実行構成」を選択する。

ii. 「Java アプリケーション」を右クリックし、「新規」を選択する。

iii. 「Java アプリケーション」直下に追加された「新規構成」を選択し、プロジェクトに“tutorial”、メイン・クラスに“jp.terasoluna.fw.batch.executor.SyncBatchExecutor”を入力する。

iv. 「引数」タブ内の「プログラムの引数」に“SMP004”を入力し、「実行」を押下する。

(2). ログの確認

コンソールに以下のログが出力されることを確認する。

デフォルトの設定では正常終了した場合、ジョブ終了コードは“0”となる。

```
[2015/**/** **:**:**] [main] [j.t.f.b.e.SyncBatchExecutor] [INFO ] [IAL025014]
SyncBatchExecutor START
・・・
```

```
[2015/**/** **:**:**] [main] [j.t.b.t.s.SMP004BLogic] [INFO ] ファイル書き込みが
正常に終了しました。
[2015/01/07 09:56:10] [main] [j.t.f.b.e.SyncBatchExecutor] [INFO ] [IAL025015]
SyncBatchExecutor END blogicStatus:[0]
```

(3). データの確認

“tutorial ¥outputFile”フォルダーの“SMP004_output.csv”のデータを確認する。入力データは日付、支店名の順でソートされており、日付、支店名が切り替わったタイミングでコントロールブレイク処理が実行される。コントロールブレイク処理では、ある取引日、ある支店名での入金回数、出金回数、入金合計、出金合計を出力する。次のような出力結果が得られる。

取引日	支店名	入金回数	出金回数	入金合計	出金合計
20111001	埼玉	2	0	1311456	0
20111001	千葉	1	1	19301	826811
20111001	東京	2	4	662349	2643052
・・・					
20111005	埼玉	2	1	1048600	203102
20111005	千葉	4	0	2128168	0
20111005	東京	2	1	999097	391908

2.6 入力が DB、出力がファイルである場合のジョブ（非同期型ジョブ実行）

- 必要な作業

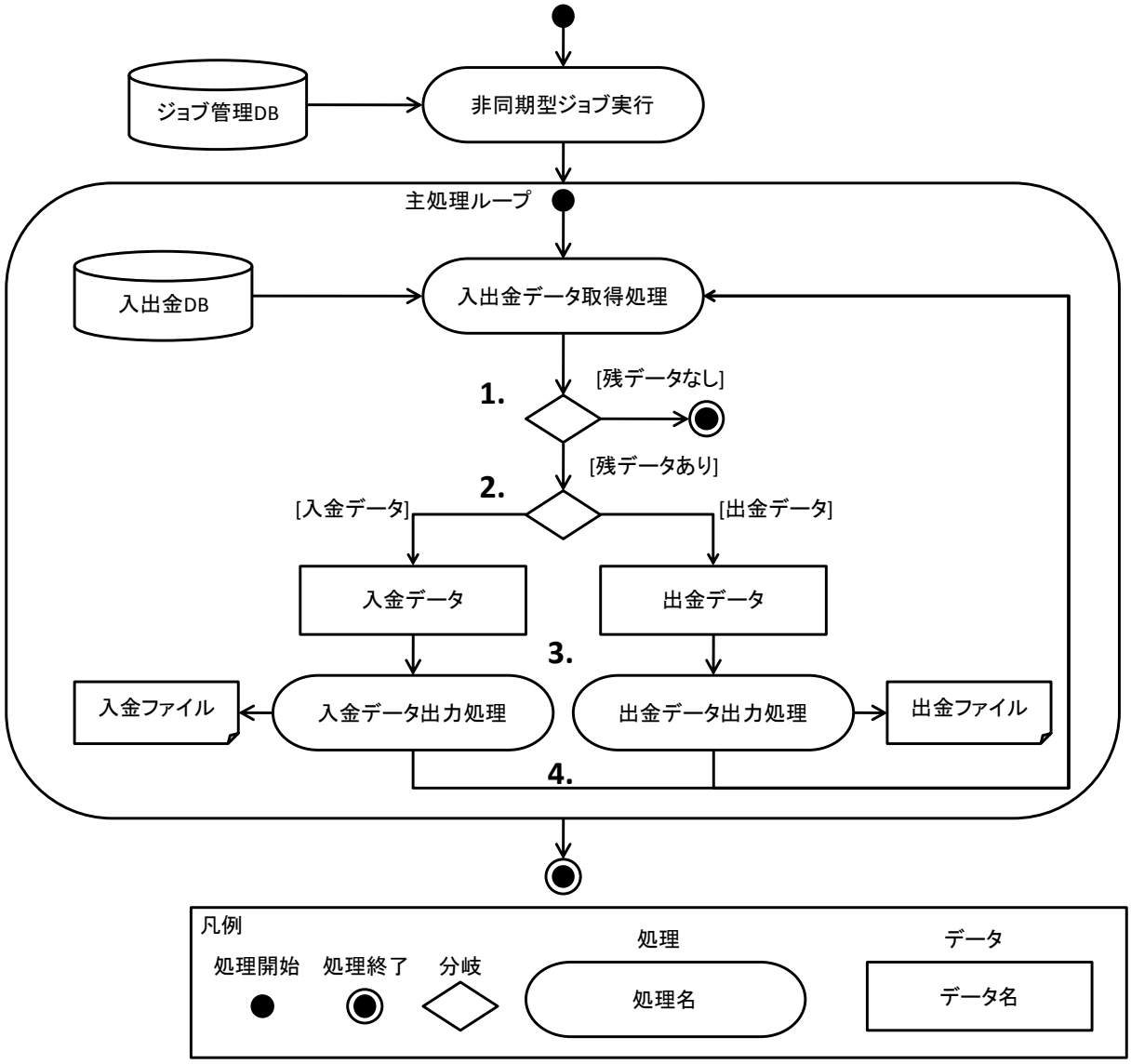
  2.6.1 定義ファイルの作成

  2.6.2 ビジネスロジックの実装

  2.6.3 起動と確認

- ジョブの内容

  図のように DB から顧客毎の入出金データを読み込み、入金データと出金データに分けてファイル出力を行うジョブを作成する。なお、本節では非同期型ジョブ実行を採用する。



1. データを取得し、存在しない場合は処理を終了する。
2. 入力パラメータの「入出金区分」を確認し、入金データと出金データに分割する。
3. 入金データは入金ファイル、出金データは出金ファイルへ出力する。
4. 1.へ戻り、次の入出金データを取得する。

- ジョブのリソース

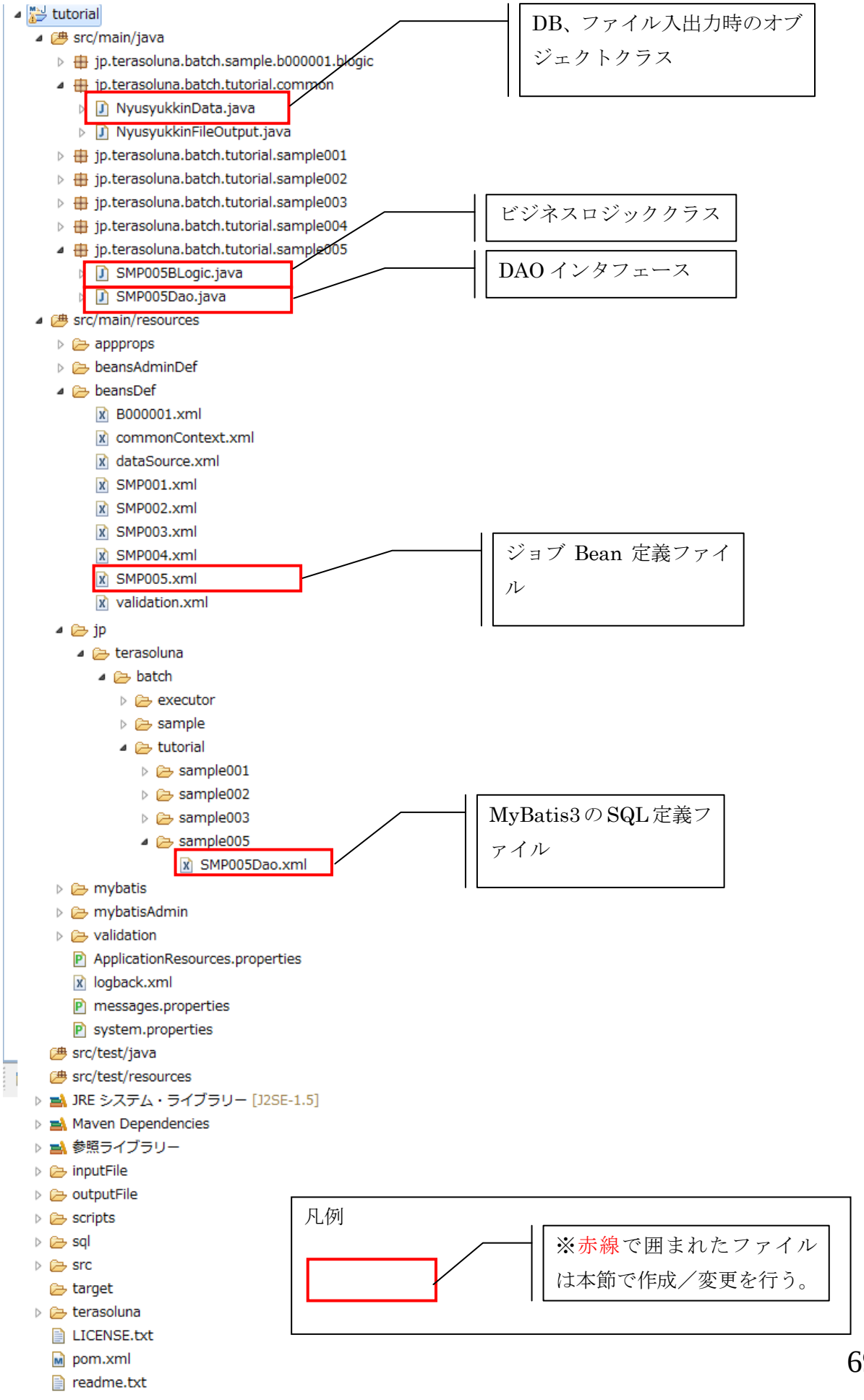
  本ジョブで作成／使用するパッケージ、ジョブ ID は以下の通りである。

  パッケージ：jp.terasoluna.batch.tutorial.common

  jp.terasoluna.batch.tutorial.sample005

  ジョブ ID： SMP005

  また、本ジョブ作成後のプロジェクトは以下のようになる。


tutorial
src/main/java
jp.terasoluna.batch.sample.b000001.blogic
jp.terasoluna.batch.tutorial.common
NyusyukkinData.java
NyusyukkinFileOutput.java
jp.terasoluna.batch.tutorial.sample001
jp.terasoluna.batch.tutorial.sample002
jp.terasoluna.batch.tutorial.sample003
jp.terasoluna.batch.tutorial.sample004
jp.terasoluna.batch.tutorial.sample005
SMP005BLogic.java
SMP005Dao.java
src/main/resources
appprops
beansAdminDef
beansDef
B000001.xml
commonContext.xml
dataSource.xml
SMP001.xml
SMP002.xml
SMP003.xml
SMP004.xml
SMP005.xml
validation.xml
jp
terasoluna
batch
executor
sample
tutorial
sample001
sample002
sample003
sample005
SMP005Dao.xml
mybatis
mybatisAdmin
validation
ApplicationResources.properties
logback.xml
messages.properties
system.properties
src/test/java
src/test/resources
JRE システム・ライブラリー [J2SE-1.5]
Maven Dependencies
参照ライブラリー
inputFile
outputFile
scripts
sql
src
target
terasoluna
LICENSE.txt
pom.xml
readme.txt
DB、ファイル入出力時のオブジェクトクラス
ビジネスロジッククラス
DAO インタフェース
ジョブ Bean 定義ファイル
MyBatis3 の SQL 定義ファイル
凡例
※赤線で囲まれたファイルは本節で作成／変更を行う。

2.6.1. 定義ファイルの作成

- ジョブ Bean 定義ファイル
- マッピングファイル

1. ジョブ Bean 定義ファイル

   [手順]

   (1). “SMP005.xml” を作成

   i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources¥beansDef” フォルダーを右クリックする。

   ii. 「新規」→「ファイル」を選択し、ファイル名に “SMP005.xml” と入力し「完了」を押下する。

   iii. “SMP005.xml” に “tutorial¥src¥main¥resources¥beansDef¥B000001.xml” の内容をコピーし、以下のように変更する。

変更前

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.sample.b000001" />
```

変更後

```
<!-- コンポーネントスキャン設定 -->
<context:component-scan base-package="jp.terasoluna.batch.tutorial.sample005" />

<!-- SMP005Dao設定 -->
<bean class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"
        value="jp.terasoluna.batch.tutorial.sample005.SMP005Dao" />
    <property name="sqlSessionTemplate" ref="sqlSessionTemplate" />
</bean>
```

2. マッピングファイル

   MyBatis3 の定義ファイルであり、SQL 定義情報を記述する。

   [手順]

   (1). “SMP005Dao.xml” を作成

   i. パッケージエクスプローラーで、“tutorial¥src¥main¥resources” フォルダーを右クリックする。

   ii. 「新規」→「ファイル」を選択し、「親フォルダーを入力または選択」に “tutorial/src/main/resources/jp/terasoluna/batch/tutorial/sample005”、ファイル名に “SMP005Dao.xml” と入力し「完了」を押下する。

   iii. “SMP005Dao.xml” を以下のように編集する。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org/DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="jp.terasoluna.batch.tutorial.sample005.SMP005Dao">
    <!-- データ取得 -->
    <!-- 入出金テーブル -->
    <select id="collectNyusyukkinData"
        resultType="jp.terasoluna.batch.tutorial.common.NyusyukkinData">
        SELECT
        SHITENNAME AS "shitenName"
        ,KOKYAKUID AS "kokyakuId"
        ,NYUSYUKKINKUBUN AS "nyusyukkinKubun"
        ,KINGAKU AS "kingaku"
        ,TORIHIKIBI AS "torihikibi"
        FROM
        NYUSYUKKINTBL
    </select>
</mapper>
```

2.6.2. ビジネスロジックの実装

本節では、ビジネスロジックの実装方法について説明する。

[手順]

(1). “NyusyukkinData.java”を作成

「2.2.2 ビジネスロジックの実装」の手順(1)を参照のこと。

(2). “SMP005Dao.java”を作成

DAO を作成する。

i. パッケージエクスプローラーで“tutorial”を右クリックする。

ii. 「新規」→「パッケージ」を選択し、名前に“jp.terasoluna.batch.tutorial.sample005”を入力し、「完了」を押下する。

iii. パッケージエクスプローラーで、“jp.terasoluna.batch.tutorial.sample005”パッケージを右クリックする。

iv. 「新規」→「インターフェース」を選択し、名前に“SMP005Dao”を入力し、「完了」を押下する。

v. “SMP005Dao.java”を以下のように作成する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample005;

import org.apache.ibatis.session.ResultHandler;

public interface SMP005Dao {

    /**
     * 入出金情報を取得する。
     * @param object SQLパラメータ引数オブジェクト
     * @param handler ResultHandler
     */
    public void collectNyusyukkinData(Object object, ResultHandler handler);
}
```

(3). “SMP005BLogic.java”を作成

ビジネスロジックを作成する。

i. パッケージエクスプローラーで、“jp.terasoluna.batch.tutorial.sample005”パッケージを右クリックする。

ii. 「新規」→「クラス」を選択し、名前に“SMP005BLogic”およびスーパークラスに“jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic”を入力し、「完了」を押下する。

iii. “SMP005BLogic.java”を以下のように作成する。

```
/*
 * Copyright (c) 2015 NTT DATA Corporation
 */
package jp.terasoluna.batch.tutorial.sample005;

import javax.inject.Inject;
import javax.inject.Named;

import jp.terasoluna.batch.tutorial.common.NyusyukkinData;
import jp.terasoluna.fw.batch.blogic.AbstractTransactionBLogic;
import jp.terasoluna.fw.batch.blogic.vo.BLogicParam;
```

```
import jp.terasoluna.fw.collector.Collector;
import jp.terasoluna.fw.collector.db.DaoCollector;
import jp.terasoluna.fw.collector.util.CollectorUtility;
import jp.terasoluna.fw.file.dao.FileLineWriter;
import jp.terasoluna.fw.file.dao.FileUpdateDAO;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Component;

/**
 * ビジネスロジッククラス。(入出金テーブルをcsvファイルに出力するクラス)
 */
@Component
public class SMP005BLogic extends AbstractTransactionBLogic {

    private static final Logger log = LoggerFactory.getLogger(SMP005BLogic.class);

    @Inject
    protected SMP005Dao dao;

    @Inject
    @Named("csvFileUpdateDAO")
    protected FileUpdateDAO csvFileUpdateDAO;

    public int doMain(BLogicParam param) {

        // ジョブ終了コード(0:正常終了、-1:異常終了)
        int returnCode = 0;

        // コレクタ
        Collector<NyusyukkinData> collector = new DaoCollector<NyusyukkinData>(
                this.dao, "collectNyusyukkinData", null);

        // ファイル出力用行ライタの取得(入金用)
        FileLineWriter<NyusyukkinData> fileLineWriterNyukin = csvFileUpdateDAO
                .execute("outputFile/SMP005_output_nyukin.csv",
```

```
                NyusyukkinData.class);

        // ファイル出力用行ライタの取得(出金用)
        FileLineWriter<NyusyukkinData> fileLineWriterSyukkin = csvFileUpdateDAO
                .execute("outputFile/SMP005_output_syukkin.csv",
                        NyusyukkinData.class);

        try {
            // DBから取得したデータを格納するオブジェクト
            NyusyukkinData inputData = null;

            while (collector.hasNext()) {
                // DBからデータを取得
                inputData = collector.next();

                // ファイルへデータを出力(1行)
                // 入出金区分により出力ファイルを変更
                if (inputData != null && inputData.getNyusyukkinKubun() == 0) {
                    fileLineWriterNyukin.printDataLine(inputData);
                }
                if (inputData != null && inputData.getNyusyukkinKubun() == 1) {
                    fileLineWriterSyukkin.printDataLine(inputData);
                }
            }
        } catch (DataAccessException e) {
            if (log.isErrorEnabled()) {
                log.error("データアクセスエラーが発生しました", e);
            }

            returnCode = -1;
        } catch (Exception e) {
            if (log.isErrorEnabled()) {
                log.error("エラーが発生しました", e);
            }

            returnCode = -1;
        } finally {
            // コレクタのクローズ
```

```
            CollectorUtility.closeQuietly(collector);

            // ファイルのクローズ
            CollectorUtility.closeQuietly(fileLineWriterNyukin);
            CollectorUtility.closeQuietly(fileLineWriterSyukkin);

            // 正常終了時のログ
            if (returnCode == 0 && log.isInfoEnabled()) {
                log.info("ファイル書き込みが正常に終了しました。");
            }
        }

        return returnCode;
    }
}
```

2.6.3.起動と確認

本節では、作成したジョブの起動と確認方法について説明する。

[手順]

(1). 非同期型ジョブの起動

i. メニューより「実行」→「実行構成」を選択する。

ii. 「Java アプリケーション」を右クリックし、「新規」を選択する。

iii. 「Java アプリケーション」直下に追加された「新規構成」を選択し、プロジェクトに“tutorial”、メイン・クラスに“jp.terasoluna.fw.batch.executor.AsyncBatchExecutor”を入力し、「実行」を押下する。

(2). ログの確認

コンソールに以下のログが出力されることを確認する。

```
[2015/**/** **:**:**] [main] [j.t.f.b.e.AsyncBatchExecutor] [INFO ] [IAL025005]
AsyncBatchExecutor START
[2015/**/** **:**:**] [main] [j.t.f.b.e.AsyncBatchExecutor] [INFO ] [IAL025006]
jobAppCd:[]
[2015/**/** **:**:**] [main] [j.t.f.b.e.c.BatchThreadPoolTaskExecutor] [INFO ]
Initializing ExecutorService  'batchTaskExecutor'
```

(3). ジョブの登録

ジョブ管理テーブルにジョブを登録するために SQL を実行する。

“terasoluna-batch-tutorial¥sql¥insert_job_data.sql”を実行すると下記のジョブが登録される。数秒後に、ジョブ管理テーブルに登録したジョブが実行される。

job_sql_id	job_app_cd	cur_app_status
nextval(’JOB_CONTROL_SEQ_001’)	SMP005	0

下記の手順で登録することができる。

① Windows のスタートメニューから[プログラム]→[PostgreSQL 9.3]→[pgAdmin III]を起動する。

② 「PostgreSQL 9.3(localhost:5432)」を選択し、右クリックメニューから「接続」を選択する。パスワード入力ウィンドウが表示されたら、「P0stgres [ﾋﾟｰｾﾞﾛｴｽﾃｨｰｼﾞｰｱｰﾙｲｰｴｽ]」と入力する。

③ 画面左のメニューから「データベース(2)」「terasoluna」を右クリックで選択し、「CREATE スクリプト」をクリックする。

④ 「SQL エディタ」のテキストフィールドに“terasoluna-batch-tutorial¥sql¥insert_job_data.sql”の内容を入力して「クエリーの実行」をクリックする。

> ※備考
> Oracle を利用する場合は、
> 「“terasoluna-batch-tutorial¥sql¥insert_job_data_Oracle.sql”」を利用する。

(4). ログの確認

コンソールに以下のログが出力されることを確認する。

```
[2015/**/** **:**:**] [j.t.f.b.e.AbstractJobBatchExecutor] [INFO ] [IAL025001]
BATCH START jobSequenceId:[*]
・・・
[2015/**/** **:**:**] [AsyncBatchExecutorThread-1-1] [j.t.b.t.s.SMP005BLogic]
[INFO ] ファイル書き込みが正常に終了しました。
[2015/**/** **:**:**] [AsyncBatchExecutorThread-1-1]
[j.t.f.b.e.AbstractJobBatchExecutor] [INFO ] [IAL025003] BATCH END
jobSequenceId:[*] blogicStatus:[0]
[2015/**/** **:**:**] [AsyncBatchExecutorThread-1-1]
[j.t.f.b.e.c.BatchThreadPoolTaskExecutor] [INFO ] Shutting down ExecutorService
'batchTaskExecutor'
```

(5). ジョブの実行完了確認

ジョブ管理テーブルを参照し、登録したジョブの実行が完了していることを確認する。

下記の手順で確認することができる。

① Windows のスタートメニューから[プログラム]→[PostgreSQL 9.3]→[pgAdmin III]を起動する。
② 「PostgreSQL 9.3(localhost:5432)」を選択し、右クリックメニューから「接続」を選択する。パスワード入力ウィンドウが表示されたら、「P0stgres [ﾋﾟｰｾﾞﾛｴｽﾃｨｰｼﾞｰｱｰﾙｲｰｴｽ]」と入力する。
③ 画面左のメニューから「データベース(2)」「terasoluna」を選択し、「スキーマ(1)」→「public」→「テーブル(2)」と選択する。
④ 「job_control」で右クリックし、「データビュー」→「先頭 100 件の表示」をクリックする。
⑤ 登録したジョブの“cur_app_status”の値が“2”であれば、ジョブの処理は終了している。

(6). データの確認

“tutorial ¥outputFile”フォルダーの“SMP005_output_nyukin.csv”および“SMP005_output_syukkin.csv”のデータを確認する。

※「2.2 入力が DB、出力がファイルである場合のジョブ(同期型ジョブ実行)」と処理内容が同じであるため、“SMP005_output_nyukin.csv”のデータは“SMP001_output_nyukin.csv”のデータと一致し、“SMP005_output_syukkin.csv”のデータは“SMP001_output_syukkin.csv”と一致する。

(4). 非同期型ジョブの停止

“C:¥tmp¥”配下に“batch_terminate_file”(拡張子なし,空ファイル)を作成する。ファイルが作成されることによってジョブが停止する。

“tutorial¥src¥main¥resources¥appprops”フォルダーの“batch.properties”に下記のような設定がある。

```
executor.endMonitoringFile=/tmp/batch_terminate_file
```

これは“C:¥tmp¥batch_terminate_file”ファイルを作成することによってジョブを停止することができることを意味する。

(注)上記ファイルは自動的に作成されないので、ジョブを停止する際に自身で作成する必要がある。

第3章　Appendix

3.1 概要

本チュートリアルの補足を説明する。各節の内容を以下に示す。

- チュートリアル学習環境の整備(Oracle)
  Oracle の設定手順について説明する。

3.2 チュートリアル学習環境の整備(Oracle)

本節では、チュートリアルを学習するための環境整備(Oracle)について説明する。

- 想定環境
  - フレームワーク： TERASOLUNA Batch Framework for Java 3.5.0
  - OS：Microsoft Windows 7 Professional
  - JDK：JDK 1.7.0_xx(x はバージョン番号)
  - データベース：Oracle Database Express Edition 12c
  - 総合開発環境：Eclipse SDK 3.7.x
    ※Pleiades All in One 日本語ディストリビューション版

- インストール/開発環境の整備

  1　アプリケーションの用意
  本資料で必要となるアプリケーションを以下に用意する。
  - JDK 1.7.0_xx(x はバージョン番号)
    ※http://www.oracle.com/technetwork/jp/java/javase/downloads/jdk7-downloads-1880260.html
  - Oracle Database Express Edition 12c (以下、Oracle)
    ※http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html
  - Eclipse SDK 3.7.x + Maven プラグイン
    ※http://mergedoc.sourceforge.jp/pleiades_distros3.7.html

  2　アプリケーションのインストール
  「2.1 チュートリアル学習環境の整備」を参照のこと。

  3　プロジェクトの準備
  「2.1 チュートリアル学習環境の整備」を参照のこと。

4　プロジェクトのインポート

「2.1 チュートリアル学習環境の整備」を参照のこと。

5　データベースの設定

5.1　Oracle を任意のディレクトリにインストールする。本チュートリアルでは、“C:¥oraclexe¥app¥oracle¥product¥12.1.0”というディレクトリにインストールしたと仮定する。

5.2　データベースを作成する。ここでは以下のように設定する。

- データベースサーバの IP アドレス：127.0.0.1
- ポート：1521
- データベース名：XE
- SID 名：XE

5.3　SQL*Plus などを利用してユーザーを作成する。ここでは以下のユーザーを作成する。

- ユーザー名：tutorial
- パスワード：tutorial
- アクセス権限：テーブル作成、データの挿入、選択、削除が行える

> ※備考
> 設定方法については Oracle のマニュアルを参照のこと。

6　入力ファイルの取得

「2.1 チュートリアル学習環境の整備」を参照のこと。

7　ファイルの出力先の作成

「2.1 チュートリアル学習環境の整備」を参照のこと。

8　“jdbc.properties の修正”

“tutorial¥src¥main¥resources¥mybatisAdmin”フォルダーと“tutorial¥src¥main¥resources¥mybatis”フォルダーにある“jdbc.properties”を以下のように書き換える。

```
#ドライバー
jdbc.driver=oracle.jdbc.OracleDriver
#URL
Jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:xe
#ユーザー名
jdbc.username=tutorial
#パスワード
jdbc.password=tutorial
```

9　“AdminDataSource.xml の修正”

この節では Oracle を使用するため、

“tutorial ¥src¥main¥resources¥beansAdminDef” フォルダーにある

“AdminDataSource.xml”のシステム利用 DAO 定義(PostgreSQL)がコメントアウトされていて、システム利用 DAO 定義(Oracle)が有効になっていることを確認する。

```
<!-- システム利用DAO定義(Oracle) -->
<bean id="systemDao" class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"
        value="jp.terasoluna.fw.batch.executor.dao.SystemOracleDao"/>
    <property name="sqlSessionFactory" ref="sysSqlSessionFactory"/>
</bean>
<!-- システム利用DAO定義(PostgreSQL)
<bean id="systemDao" class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"
        value="jp.terasoluna.fw.batch.executor.dao.SystemPostgreSQLDao"/>
    <property name="sqlSessionFactory" ref="sysSqlSessionFactory"/>
</bean>
-->
```

- データベースの初期化手順
  チュートリアル完成版プロジェクト“terasoluna-batch-tutorial”では以下のデータベースの初期化スクリプトとバッチファイルを提供する。

スクリプト／バッチファイル	説明
terasoluna-batch-tutorial¥sql¥oracle¥create_sequence_job_control.sql	チュートリアルで使用するジョブ管理シーケンスを作成する。
terasoluna-batch-tutorial¥sql¥oracle¥create_table_job_control.sql	チュートリアルで使用するジョブ管理テーブルを作成する。初期値は挿入されない。
terasoluna-batch-tutorial¥sql¥oracle¥create_table_nyusyukkin.sql	チュートリアルで使用する入出金テーブルを作成する。初期値は挿入されない。
terasoluna-batch-tutorial¥sql¥oracle¥drop_all_sequence.sql	チュートリアルで使用するジョブ管理シーケンスを削除する。
terasoluna-batch-tutorial¥sql¥oracle¥drop_all_tables.sql	チュートリアルで使用する全テーブルを削除する。
terasoluna-batch-tutorial¥sql¥oracle¥insert_all_data.sql	チュートリアルで使用するテーブルに初期値を挿入する。
terasoluna-batch-tutorial¥sql¥oracle¥terasoluna_tutorial_batch.sql	上記の SQL をすべて実行し、チュートリアルアプリケーションを実行可能な初期状態を作成する。
terasoluna-batch-tutorial¥sql¥oracle¥setup_for_Oracle.bat	“terasoluna_tutorial_batch.sql”を実行するバッチファイル。

- “setup_for_Oracle.bat”を実行してデータベースをチュートリアルアプリケーション実行可能な初期状態とする。

- JDBC のクラスパス設定
  Eclipse から Oracle の JDBC へのクラスパス設定を行う。
  (1) “tutorial”を右クリックし、「プロパティー」を選択する。
  (2) 「Java のビルド・パス」を選択し、「ライブラリー」タブを押下する。
  (3) 「外部 JAR の追加」を押下し、
  “C:¥oraclexe¥app¥oracle¥product¥12.1.0¥jdbc¥lib”にある“ojdbc7.jar”を選択する。

  ※備考
  “ojdbc7.jar”が存在しない場合は Oracle 社の WEB サイト
  “http://www.oracle.com/technetwork/database/features/jdbc/jdbc-drivers-12c-download-1958347.html”から入手する。

  (4) 「OK」を押下する。

TERASOLUNA Batch Framework for Java 3.5.0
チュートリアルマニュアル 第 1.0.0 版

2015 年 2 月　　第 1.0.0 版発行

発行者　　技術開発本部 ソフトウェア工学推進センタ

〒135-8671
東京都江東区豊洲 3-3-9
豊洲センタービルアネックス 12 階
Tel: 050-5546-2482
Fax: 03-3536-3007