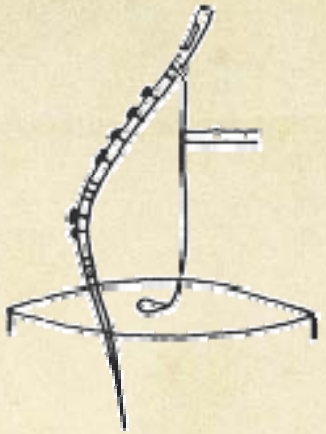
現場ソリューション2
DBFlute

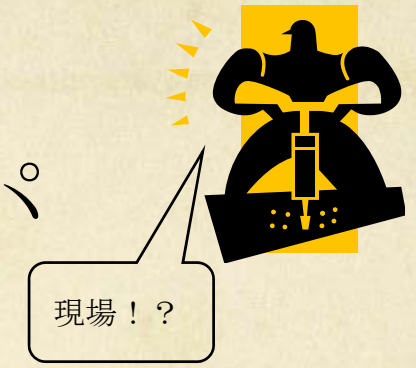


久保　雅彦

Seasar Conference 2008 Autumn

DBFluteとは？

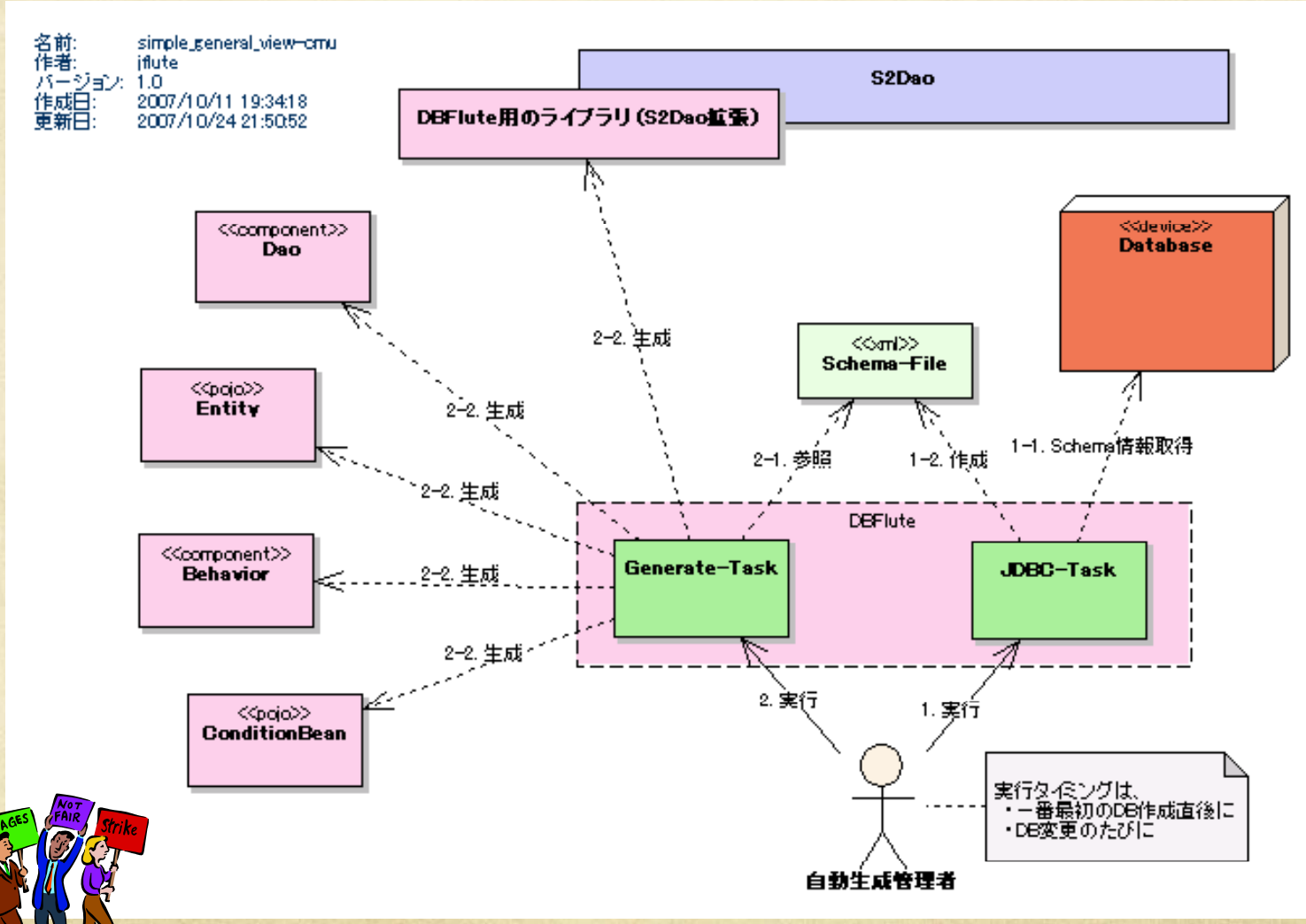
現場指向のO/Rマッパ



- 実際の開発現場で鍛えられつつ作り上げられてきたため、現場にフィットすることを重視しています。
- Apache TorqueとS2Daoの良い部分を取り込み、独自のスタイルを提供します。

  (S2Daoの自動生成ツールでもある)

DBFlute抽象概念図



前回の現場ソリューション

- CBのDB変更耐久性
- 区分値の解決
- 共通カラムの自動設定
- ページングナビゲーション
- one-to-many
- カーソル検索
- などなど

今回の現場ソリューション

- 外だしSQL(2WaySQL)のDB変更耐久性
- DDLやテストデータの管理
- 別言語・別DIコンテナ
- （今後の展望）

[1]2Wayテスト:悩み

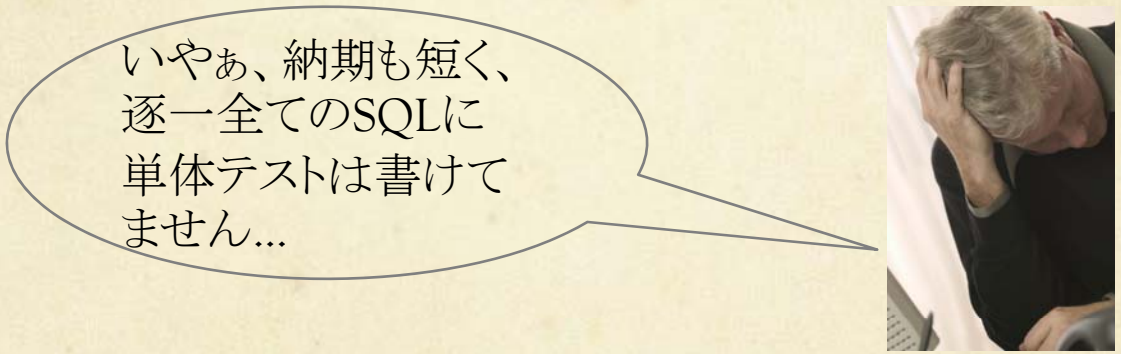
- DB変更時に外だしSQLはやっぱりつらい。。。

2Wayと言えども結局タイプ
セーフじゃないから...

ちなみに2WayじゃないO/Rマッパ
でもSQL書いてれば同じ話です

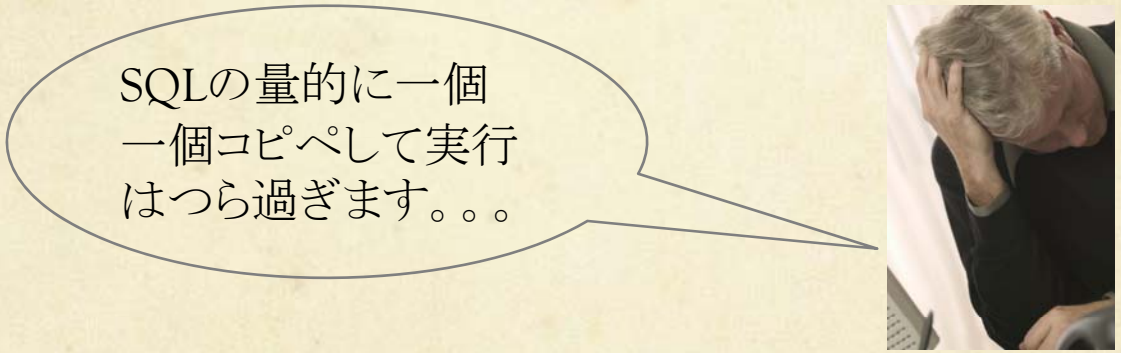
[1]2Wayテスト:悩み疑問1

- 全部単体テストを書いたらどうですか？



[1]2Wayテスト:悩み疑問2

- 2WaySQLだったら一個一個流したら？



[1]2Wayテスト:DBFluteなら

DBFluteなら...

[1]2Wayテスト:OutsideSqlTest

OutsideSqlTest!

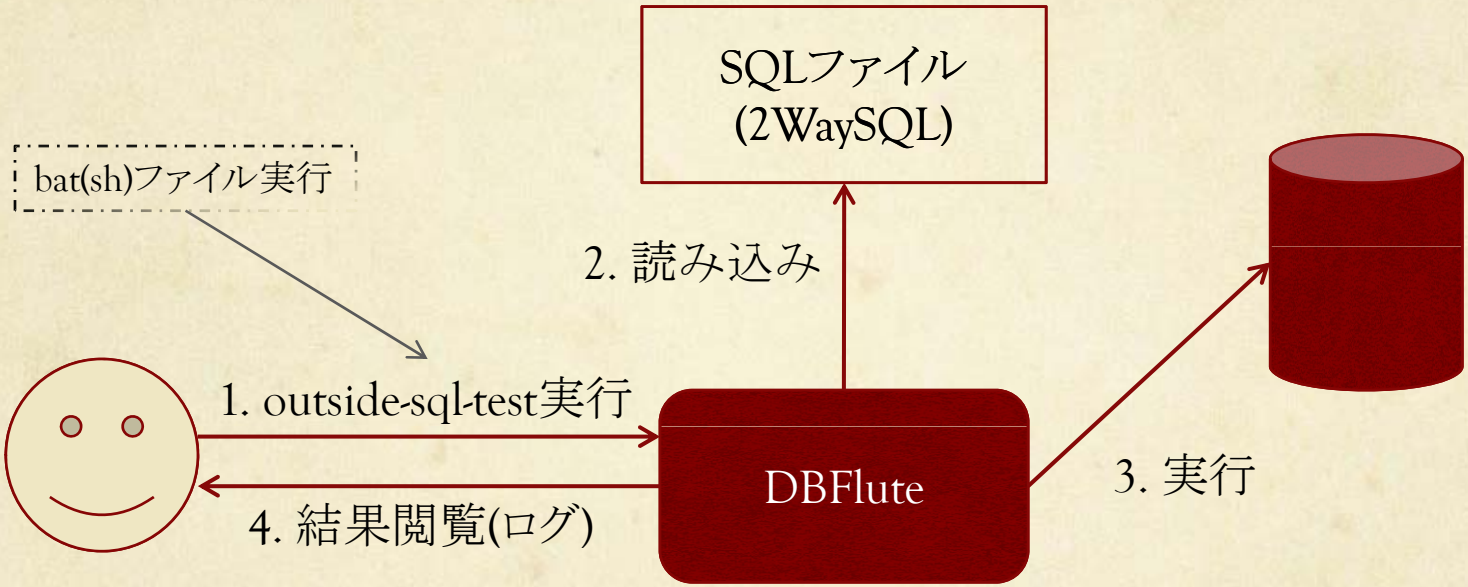
[1]2Wayテスト:
OutsideSqlTestとは？

外だしSQLを2WaySQLとして

実際のDBに一括実行して

テストするDBFluteのタスク

[1]2Wayテスト: OutsideSqlTest概念図



[1]2Wayテスト: OutsideSqlTestのメリット

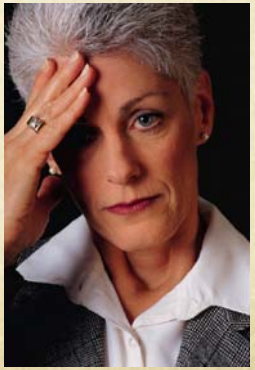
- DBAがDB変更時の影響を一気に確認できる
- プログラマがDBツールなしで簡単実行
- Sql2Entity対象外のSQLもチェック可能

地味ながら実際の現場でとても役立つ

[2]DDL&データ:悩み

- DDLやテストデータの管理がごちゃごちゃ

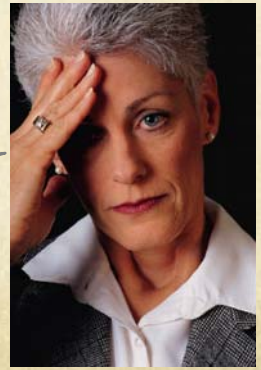
プログラマへのDDLの配布
&フェーズ毎のテストデータ管理
などなどもっとスマートに安全にしたい



[2]DDL&データ:悩み詳細1

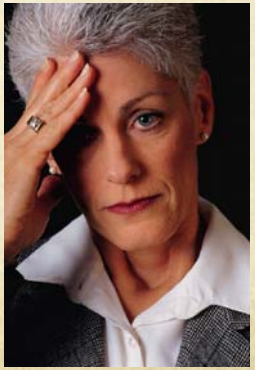
- スクリプト作成は意外にコスト
  - DDL実行しても古いテーブル残ったりしませんか？
  - データがInsert文のみなら簡単だがメンテが大変
    - エクセル管理が良いのだが...
  - データ登録のエラーハンドリングは大切でも大変
  - データのフェーズ毎の管理をするとさらに大変

スクリプト作成担当者が、プログラマからの
フィードバックを都度都度メンテしてて、
本来の仕事が進まないケースがあったの



[2]DDL&データ:悩み詳細2

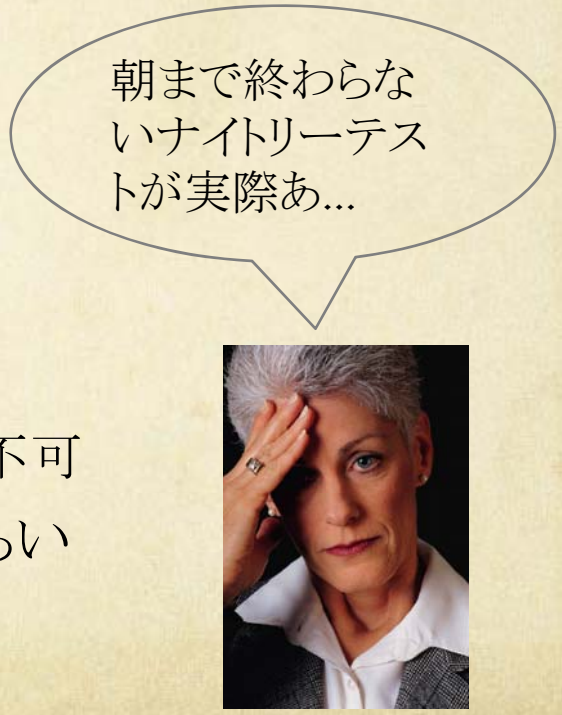
- 共通カラムのテストデータを作るの大変
  - 共通カラムのテストデータの大抵の場合固定でOK
- エクセルのシート名の32文字問題
- ストアドも実行したい
- テストデータの整合性チェックもしたい



色々言うとスクリプト作成担
当者がまた泣く...

[2]DDL&データ:悩み詳細3

- テストケース毎のエクセルテストデータは実はつらい
  - DB変更に弱い
    - ちらばるテストデータのテーブル定義
    - 直しても直しても終わらない
  - テスト実行が遅い
    - テスト実行に対する抵抗感が芽生える
    - ローカルでのデグレ防止一括テストが不可
  - DBAがテストデータのレビューがしづらい



[2]DDL&データ:DBFluteなら

DBFluteなら...

[2]DDL＆データ:ReplaceSchema

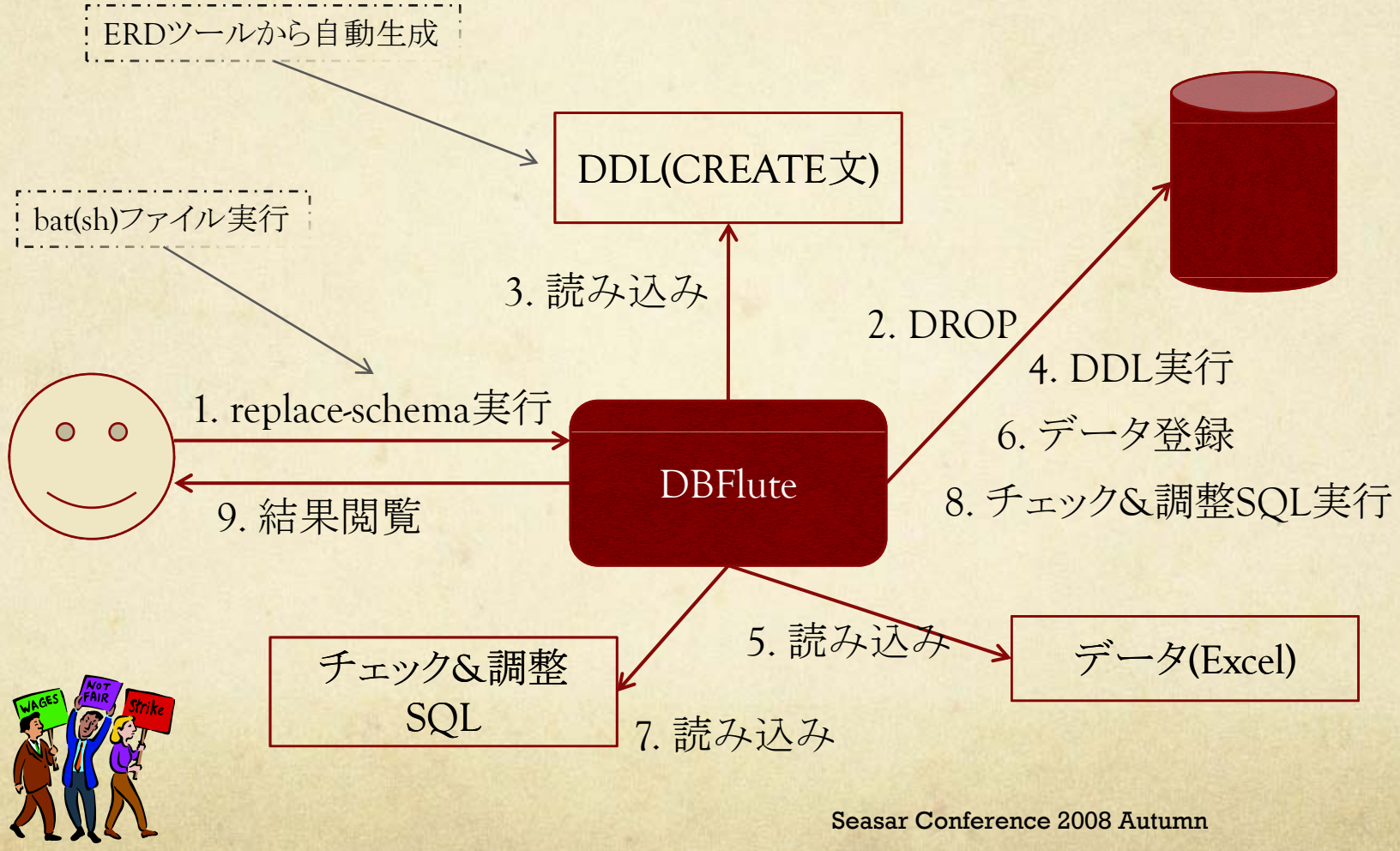
ReplaceSchema!

[2]DDL&データ: ReplaceSchemaとは？

スキーマを作り直すDBFluteのタスク

- スキーマ初期化&作成
- データ登録
- データチェック&調整

[2]DDL&データ: ReplaceSchema概念図



[2]DDL&データ:
スキーマ初期化&作成の概要

Dropの時間短縮のため

1. 既存テーブルをtruncate (できるものだけ)
2. 既存テーブルのFKを全てDrop
3. 既存テーブルを全てDrop

DBのメタデータより自動で発行

4. DBFluteクライアント/playsql/replace-schema*.sqlのSQLを実行
   - 「replace-schema」で始まる「.sql」ファイルをアスキー順で読み込む
   - セミコロン「;」区切りで順番にSQLを実行
   - 「~ #df:begin#」&「~ #df:end」で囲うことにより、セミコロン「;」を区切りとせず、ひとまとまりのSQLとすることができる。
   - 途中でSQLが例外になったらWarningログが出力されて続行
   - SQLの最後の「/」は除去される(SQL*Plusなどとの相性)
   - ファイルエンコーディングはUTF-8

主にストアドのCREATE文

DDL文はERDツールから生成するのがベスト
(そのときFKやテーブルのDROP文は不要)

[2]DDL&データ:
スキーマの初期化&作成の特徴

- 古いテーブルが残らない
  - 古いテーブル用の一時的なDrop文は不要
- 実行ログが残る
  - DBFluteクライアント/logs/dbflute.log
- DB変更があっても各開発者の環境への反映が容易

DDLやDBFlute自体をバージョン管理にし、各開発者はチェックアウト(もしくは更新)して実行するだけでDB環境が最新に！

[2]DDL&データ:
データ登録の概要

1. /playsql/data/common配下のデータをDBに登録
   - dataLoadingTypeに関わらず必ず最初に登録される。

   単体テストでも結合テストでも本番でも変わらないマスタデータなど

2. /playsql/data/xxx配下のデータをDBに登録
   - デフォルトは/playsql/data/ut配下
   - xxx部分はdataLoadingTypeで指定する(後述)。

   テストのフェーズごとに変わるデータなど

dataLoadingType: などデータが利用されるフェーズ(「ut」や「it」)の種別

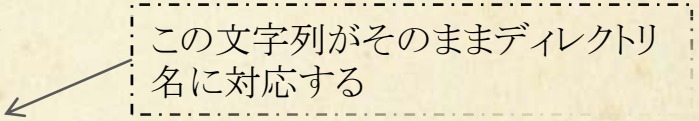
[2]DDL&データ:
データ登録のデータ切り替え

- dataLoadingTypeを指定することで登録データを切り替える

  /playsql/data/[dataLoadingType]/...

1. dfprop/replaceSchemaDefinitionMap.dfpropを作成
2. dataLoadingTypeを指定

```
map:{
    ; dataLoadingType = it
    ; ...
}
```



[2]DDL&データ:
tips:環境ごとの設定の切り替え

- dfpropファイルを環境毎に切り替える

1. 環境変数「DBFLUTE_ENVIRONMENT_TYPE = abc」を定義
2. dfprop/abcディレクトリを作成
3. dfprop/abc/配下のdfpropファイルが優先される
   (該当dfpropファイルがabc配下になければdfprop配下をみる)

この方法でdatabaseInfoMap.dfpropやreplaceSchemaDefinitionMap.dfpropを切り替えて、「Abcサーバでは、どのDBにどのdataLoadingTypeでReplaceSchemaする」を実現する。PCの環境変数に定義してもいいし、「.bat(.sh)」で定義してコマンド毎に切り替えられるようにしても良い。(DBFlute-0.7.9より)

[2]DDL&データ:
データ登録のエクセルデータ

- /playsql/data/ut/xls配下の「.xls」ファイルが対象
- アスキー順に読み込んで実行
  - ex. 10-abc.xls, 20-def.xls
  - データ登録失敗したらエラーログを出力してその時点で中断

先頭に番号付けると順序がわかりやすいのでお奨め

- シート名はテーブル名
  - 左から順番に読み込んで実行
  - 「#」で始まるシートは対象外(コメント用シートなど)
  - 除外シートを別途正規表現で指定可能(skipSheet)
  - シート名30文字問題に対応(後述:table-name.txt)

基本はS2Unitのエクセルと同じ仕様

- 一行目はカラム定義
  - 固定値で良い共通カラムは不要(後述:default-value.txt)
- 二行目以降はデータ
  - それぞれのデータはトリムされる
    - トリムしないカラムは指定可能(not-trim-column.txt)

セルのタイプを全て文字列にしてからデータを記述するのがお奨め(エクセルのクセを回避)

[2]DDL&データ:
データ登録の共通カラム解決

- 固定値で良い共通カラムは一括設定

ファイルエンコーディングはUTF-8

1. データファイルと同じディレクトリに「default-value.txt」を作成
2. 「カラム = 固定値」の形で設定

```
map:{
    ; REGISTER_DATETIME = sysdate
    ; REGISTER_USER = replace-schema
    ; …
}
```

実行時の現在日時が登録される

「replace-schema」という値が登録される

3. データにカラム定義がなくても設定の固定値が登録される

データに定義されていればそちらが優先される

[2]DDL&データ:
データ登録の30文字問題解決

- シート名に入りきらないテーブル名を解決

ファイルエンコーディングはUTF-8

1. データファイルと同じディレクトリに「table-name.txt」を作成
2. 「(任意の)省略テーブル名= テーブル名」の形で設定

```
map:{
    ; PRO_SERV_DTL = PRODUCT_SERVICE_DETAIL
    ; ...
}
```

3. シート名の先頭に「$」を付けて省略テーブル名を指定

シート名 = $PRO_SERV_DTL

4. 省略テーブル名を解決して実行されるようになる

[2]DDL&データ:
データ登録の特徴

- フェーズ毎のデータ管理
- データが編集しやすいエクセル管理
- テストデータの一元管理(単体テストのスピード向上)
- 実行ログが残る
  - DBFluteクライアント/logs/dbflute.log
- 共通カラムは一括設定
- シート名に入らないテーブルに対応

[2]DDL&データ:
データチェック&調整の概要

1. /playsql/take-finally*.sqlのSQLを実行
   - 基本仕様はreplace-schema*.sqlと同じ
   - データが登録された後に実行される
     - 暗号化やアナライズなどデータの微調整が可能
     - データの整合性チェックが可能(業務的な制約など)

【カウントが0でなければ例外】
```
-- #df:assertCountZero#
select count(*)
  from ABC
 where ...
;
```

【件数が0でなければ例外】
```
-- #df:assertListZero#
select ABC,_ID DEF
  from ABC
 where ...
;
```

[2]DDL&データ:
データチェック&調整の特徴

- データ登録後の調整が可能
- データの不整合チェックが可能
  - 不整合なテストデータでのテストを未然に防ぐ
  - 「データが悪いのかプログラムが悪いのか？」の迷いが生じるデータバグを防ぐ

[2]DDL&データ: ReplaceSchemaの思想

テストデータの管理・運用が

プロジェクトの成否に大きく関わる

そのために
- スキーマを「DDL・データ・チェック」としてワンセット管理
- DB変更時のテストデータの修正し易さ・見通しの良さ
- テストデータ作成者の手間の削減
- 確実な各開発者の環境へのDB反映
- どのプロジェクトに行っても同じやり方で環境構築

などなど

Jiemamyの思想に通じるところがある

[3]別なんとか:悩み

- 別プロジェクトにいくと別言語・別DIコンテナだったりして、プロジェクト間の開発者の行き来がしづらい

私がマネージャだったら、
効率の良い人事に悩む
わね、きっと

[3]別なんとか:悩み詳細

- C#の仕事が増えてきた
  - 言語が変わればフレームワークも変わる
- Spring Frameworkが前提の仕事もある
  - DIコンテナが変わればフレームワークも変わる

別のプロジェクトに行きたくない。。。
残して！今のプロジェクトに残してー！

[3]別なんとか:DBFluteなら

DBFluteなら...

[3]別なんとか:別言語・別コンテナ

C#版のDBFlute!

Springのの版DBFlute!

[3]別なんとか:C#版

- C#-3.0以降対応
- S2Container.NETの「Quill」を利用
- dbflute-nbasic-exampleにてExampleも充実

[3]別なんとか:Spring版(Java)

- Spring Framework-2.5(Java)以降対応
- S2Dao-Springを利用
  - 正式リリースされていないが正常に動作する
- dbflute-spring-exampleにてExampleも充実

DBFlute今後の展望

1.0へ

Emechaの充実
- 新規作成ウィザードの充実(0.5)
- バージョンアップウィザード(0.5)
- タスクの実行メニュー(0.5)

ドキュメント整備
- ドキュメント設計(0.5)
- ドキュメント作成(2.5)

Example/Test環境整備
- C#版のDB毎のExample
- Java版Exampleの整理整頓

モジュール自体のメンテ
- C#版の不足機能実装(1.0)
- Java版の安定化
- C#版の安定化

※大きなタスクはあまり残っていない！

最後

ご清聴ありがとうございました