Seasar Conference 2007 Spring

Kuina-Dao入門

2007.05.27

中村年宏

自己紹介

- 名前：中村年宏
  - ブログ：http://d.hatena.ne.jp/taedium/

- コミッタとして参加しているプロジェクト
  - S2Container
  - Kuina-Dao
  - S2Hibernate
  - S2TopLink (sandboxプロジェクト)
  - S2OpenJPA (sandboxプロジェクト)
  - S2Cayenne (sandboxプロジェクト)

アジェンダ

- Kuina-DaoとJPA
- Kuina-DaoとS2Dao
- Kuina-Daoの使いどころ
- Kuina-Daoの使いどころ アドバンスド
- ロードマップ

Kuina-DaoとJPA

JPAとは

- Javaの標準O/Rマッピング仕様
  - Java Persistence API
  - Java SE環境で利用可能
- 代表的なJPA実装プロダクト
  - Hibernate
  - TopLink Essentials
- 特徴
  - Java SE 5.0が必須
  - アノテーションによるマッピング
  - 永続コンテキスト
  - 標準
    - ツールのサポートを受けやすい
    - 情報量が豊富
    - 導入の障壁が低い

Kuina-Daoとは

- Java Persistence API(JPA)を使いやすくするためのDAOフレームワーク
  - 「くいなだお」
  - http://kuina.seasar.org/ja/index.html
  - 現在のバージョンは1.0
- JPAの実装であるHibernateやTopLink Essentialsと組み合わせて使う
  - 今後はOpenJPAやCayenneとの連携もサポート予定
  - Kuina-Coreは現在構想中
- メリット
  - コードの削減

コード例

Kuina-Daoを利用する場合

```
public interface EmpDao {
    @Orderby(“empNo”)
    Emp findByEmpNo(Integer empNo_GE);
}
```

Kuina-Daoを利用しない場合（JPAを直接利用する場合）

```
public class EmpDao {

    @PersistenceContext
    private EntityManager em;

    public Emp findByEmpNo(Integer empNo) {
        return (Emp)em.createQuery(“SELECT e FROM Emp e WHERE e.empNo >= :empNo”
ORDER BY empNo)
        .setParameter(" empNo ", empNo)
        .getSingleResult();
    }
}
```

コード例

Kuina-Daoを利用する場合

```
public interface EmpDao {
    @Orderby(“empNo”)
    List<Emp> findByEmpNo(Integer empNo_GE);
}
```

Kuina-Daoを利用しない場合（JPAを直接利用する場合）

```
public class EmpDao {

    @PersistenceContext
    private EntityManager em;

    public List<Emp> findByEmpNo(Integer empNo) {
        return (Emp)em.createQuery(“SELECT e FROM Emp e WHERE e.empNo >= :empNo”
ORDER BY empNo)
        .setParameter(" empNo ", empNo)
        .getResultList();
    }
}
```

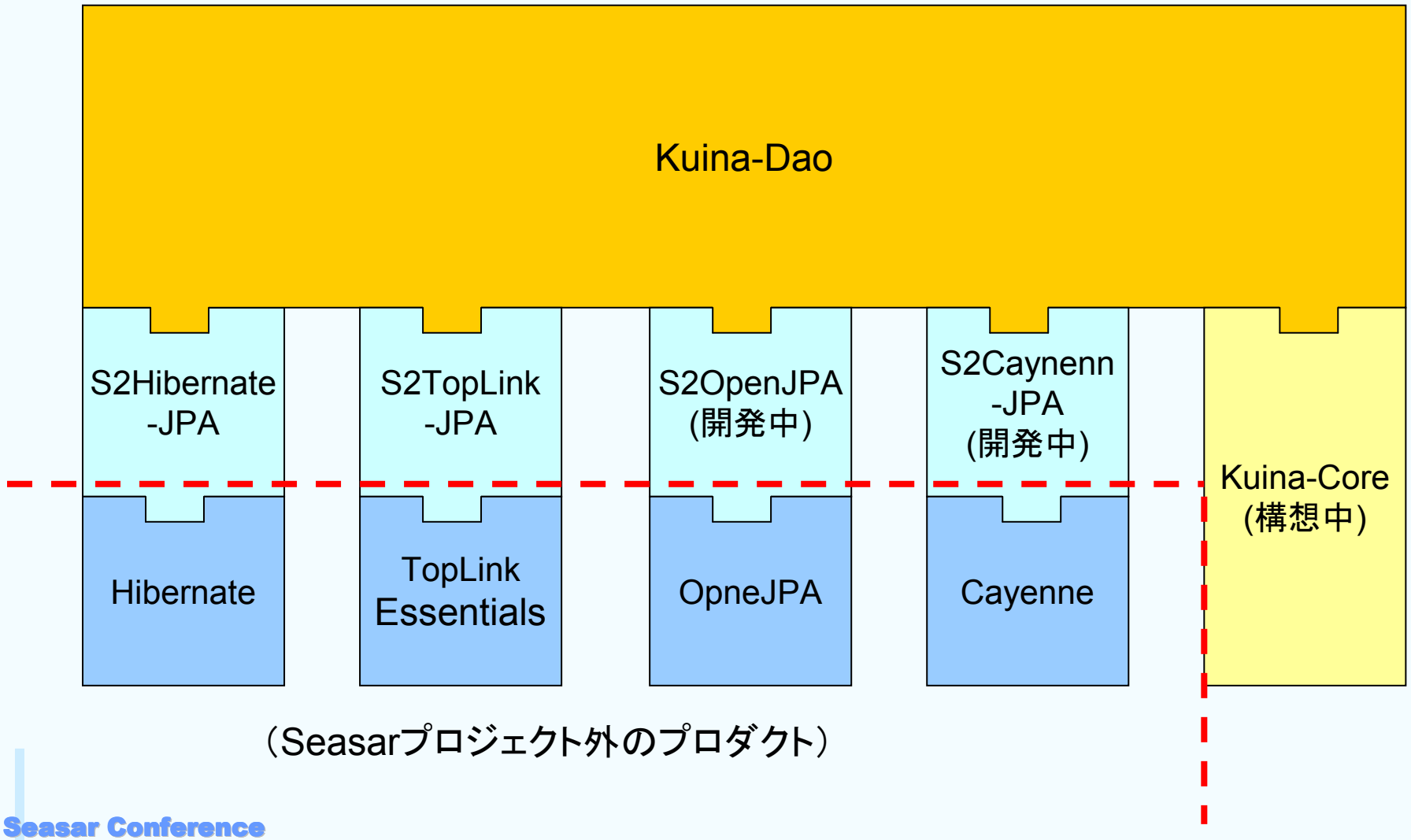
Kuina-Daoの特徴

1. 必要最小限のJavaコード
   - インターフェースだけが必要で実装クラスは不要
     - 抽象クラスを用意すれば独自ロジックを書ける
   - メソッド名やパラメータ名の命名規約を利用
   - 必要に応じてアノテーションを使用可

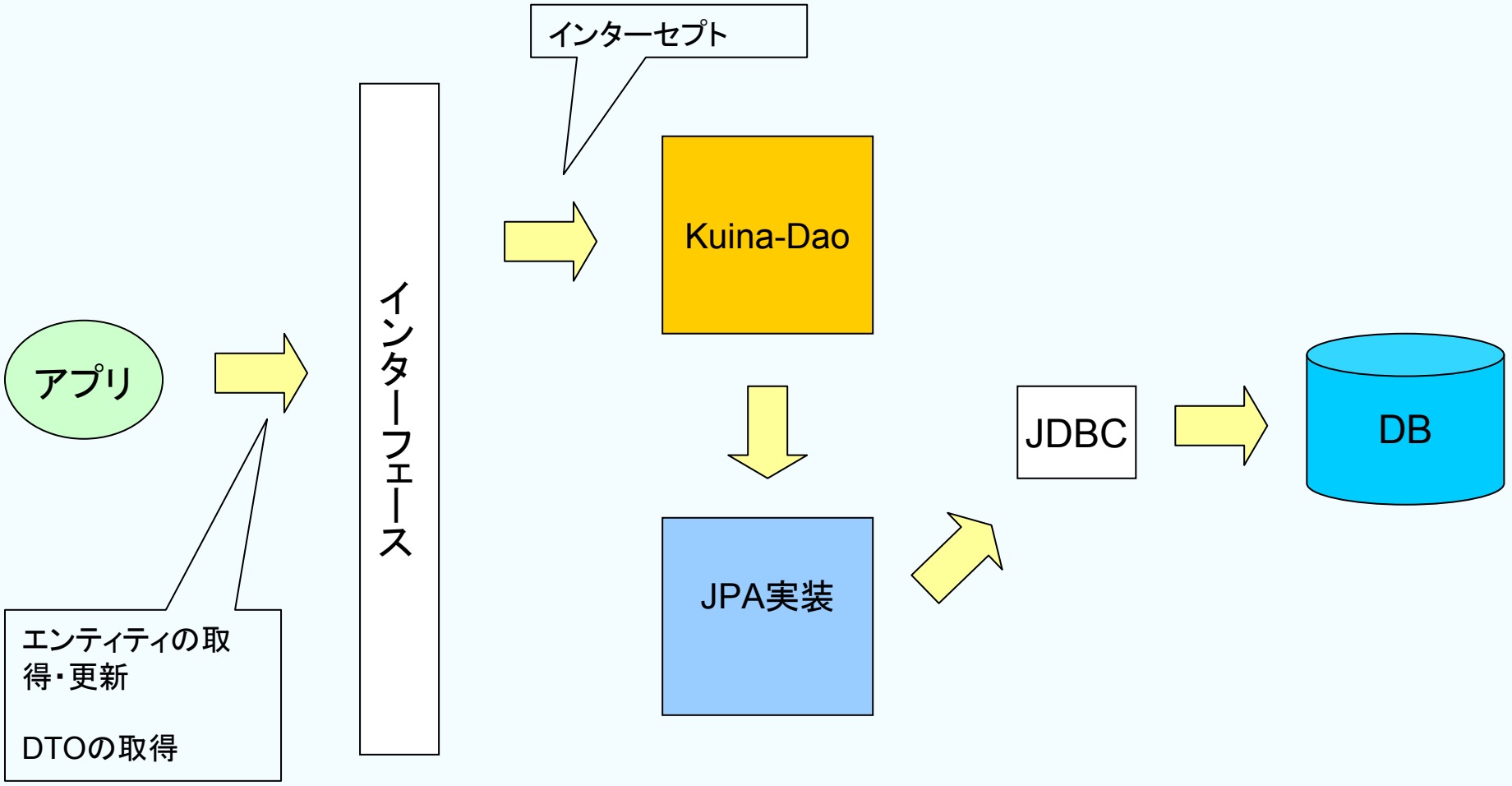
2. 問い合わせを簡略化
   - JPQLの自動生成
     - 動的な問い合わせに対応
   - SQLの動的な問い合わせに対応

3. パラメータ名をKuina-Daoで利用するためにDiiguを使用
   - DiiguはAnt、Maven、Eclipse pluginで利用可能

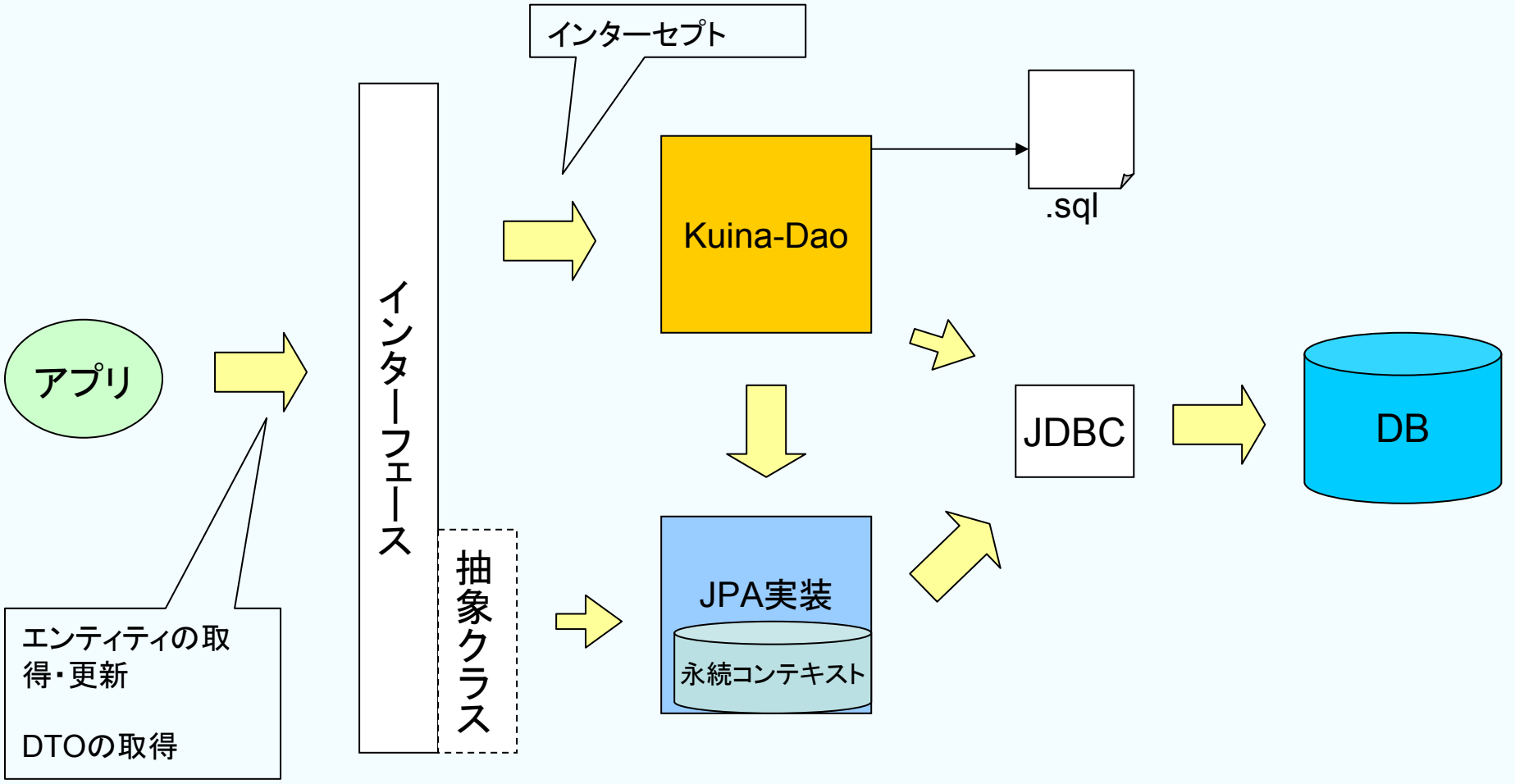
Kuina-Daoの位置づけ



Kuina-Daoの仕組み – 簡略版



Kuina-Daoの仕組み – 詳細版

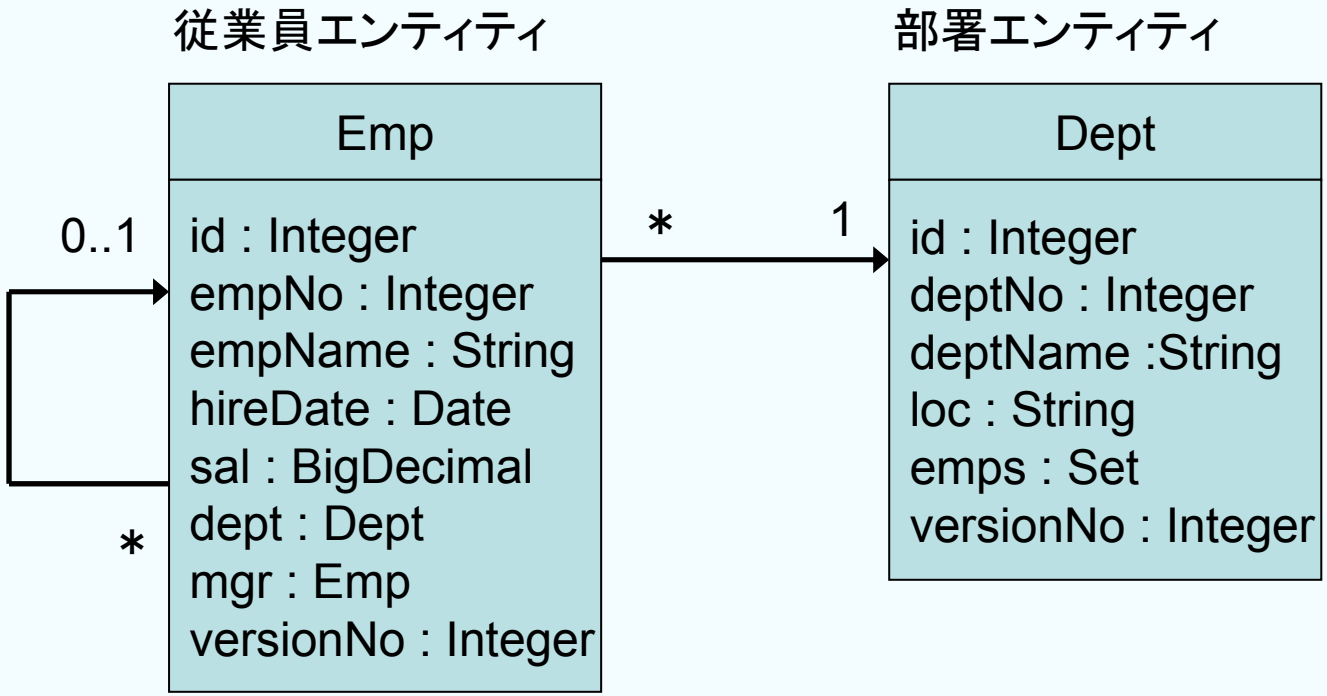


デモ

- Kuina-Daoを使った問い合わせ
  - 引数を条件とする検索
  - エンティティのプロパティを条件とする検索
  - DTOのプロパティを条件とする検索
  - JPQLによる検索
  - SQLによる検索
  - Criteriaによる検索

デモ - エンティティクラスの構成

- エンティティクラスの構成



デモ - 引数を条件とする検索

1. 引数名は検索対象のプロパティ名 + 検索条件を表すサフィックスとする。

```
@Orderby("id")
public List<Emp> findBySal(BigDecimal sal_GE);
```

実行されるJPQL

```
SELECT emp FROM Emp AS emp WHERE (emp.sal >= :sal_GE) ORDER BY emp.id
```

2.関連先のエンティティのプロパティを検索条件に含めることもできる。

```
@Orderby("id")
public List<Emp> findBySalDeptName(BigDecimal sal_GE, String dept$deptName);
```

実行されるJPQL

```
SELECT emp FROM Emp AS emp INNER JOIN emp.dept AS dept WHERE ((emp.sal
>= :sal_GE) AND (dept.deptName = :dept$deptName)) ORDER BY emp.id
```

3.INを使った検索条件。

```
@Orderby("id")
public List<Emp> findByEmpNoArray(Integer[] empNo_IN);
```

実行されるJPQL

```
SELECT emp FROM Emp AS emp WHERE emp.empNo IN
(:empNo_IN0, :empNo_IN1, :empNo_IN2, :empNo_IN3, :empNo_IN4) ORDER BY emp.id
```

デモ - エンティティのプロパティを条件とする検索

引数には検索条件を設定したエンティティを受ける。

```
@Orderby("id")
public List<Emp> findByExample(Emp emp);
```

```
Emp emp = new Emp();
emp.setEmpNo(empNo);
emp.setEmpName(empName);
emp.setHiredate(hiredate);
emp.setSal(sal);

Emp mgr = new Emp();
mgr.setEmpName(mgrName);
emp.setMgr(mgr);

Dept dept = new Dept();
dept.setDeptName(deptName);
emp.setDept(dept);

empDao.findByExample(Emp emp);
```

使用例

実行されるJPQL

```
SELECT emp FROM Emp AS emp INNER JOIN emp.dept AS dept WHERE ((emp.empName
= :empName) AND (emp.sal = :sal) AND (dept.deptName = :dept$deptName)) ORDER BY emp.id
```

デモ - DTOのプロパティを条件とする検索

引数には検索条件を設定したDTOを受ける。

```
@Orderby("id")
public List<Emp> findByDto(EmpConditionDto dto);
```

```
public class EmpConditionDto {
        private String empName_STARTS;
        private BigDecimal sal_GE;
        private String dept$deptName_CONTAINS;
        // getter, setter
}
```

DTOを定義。
プロパティ名はエンティティのプロパティ名 + 検索条件を表すサフィックスとする。

```
EmpConditionDto dto = new EmpConditionDto();
dto.setEmpName_STARTS(empName);
dto.setSal_GE(sal);
dto.setDept$deptName_CONTAINS(deptName);

empDao.findByDto(EmpConditionDto dto);
```

使用例

実行されるJPQL

```
SELECT emp FROM Emp AS emp INNER JOIN emp.dept AS dept WHERE ((dept.deptName LIKE :dept$deptName_CONTAINS) AND (emp.empName LIKE :empName_STARTS) AND (emp.sal >= :sal_GE)) ORDER BY emp.id
```

デモ - JPQLによる検索

名前つきクエリを呼び出す。

```
public List<Emp> findByEmpName(String empName);
```

```
<named-query name="EmpDao.findByEmpName">
<query>
        SELECT e FROM Emp e INNER JOIN e.mgr m
        WHERE
          e.empName = :empName
          OR
          m.empName = :empName
        ORDER BY e.id
</query>
</named-query>
```

EmpOrm.xmlに名前つきクエリを定義しておく

デモ - SQLによる検索

外部ファイルに定義したSQLをDTOにマッピングする。

```
public List<SalSumDto> getSalSum();
```

```
public class SalSumDto {
    private String deptName;
    private BigDecimal sal;
}
```

DTOを定義。

```
SELECT
    d.DEPT_NAME deptName,
    SUM(e.SAL) sal
FROM
    Dept d INNER JOIN Emp e ON d.ID = e.DEPT_ID
GROUP BY
    d.DEPT_NAME
ORDER BY
    d.ID
```

EmpDao_getSalSum.sqlにSQLを定義しておく

デモ - Criteriaによる検索

外部ファイルに定義したSQLをDTOにマッピングする。

```
public List<Emp> findBySal(BigDecimal from, BigDecimal to);
```

```
import static org.seasar.kuina.dao.criteria.CriteriaOperations.*;

public abstract class Emp2DaoImpl implements Emp2Dao {

        @PersistenceContext
        private EntityManager em;

        public List<Emp> findBySal(BigDecimal from, BigDecimal to) {
                return select().from(Emp.class,"e").where(between("e.sal",from,to))
                .orderby("e.id").getResultList(em);
        }
}
```

Javaクラスでクエリを組み立てる

実行されるJPQL

```
SELECT e FROM Emp AS e WHERE (e.sal BETWEEN 1000 AND 2000) ORDER BY e.id
```

Kuina-DaoとJPA – まとめ

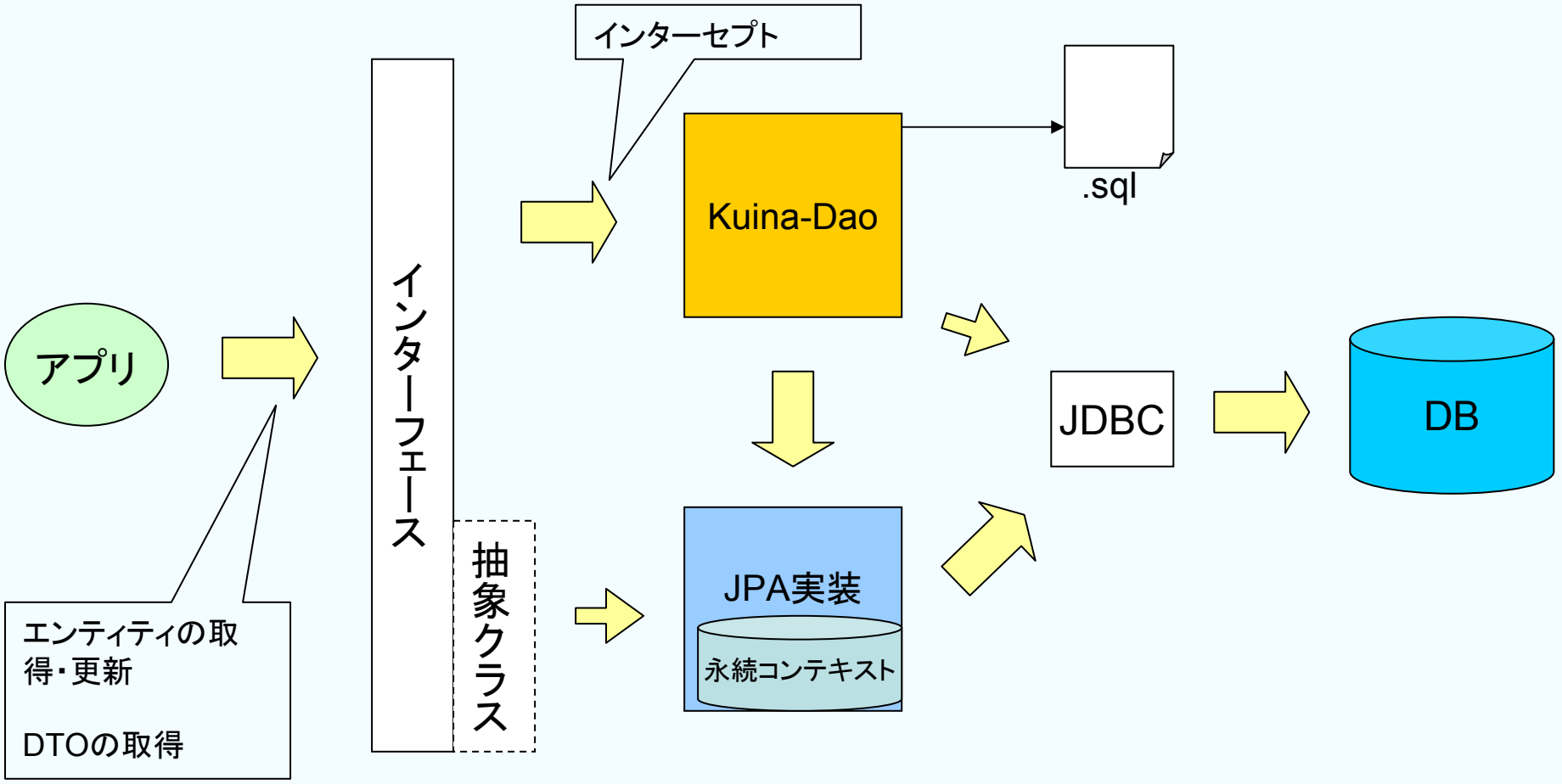
		Kuina-Dao + JPA実装	JPA実装を直接利用
1	静的な問い合わせ	○	△
2	動的な問い合わせ	○	×
3	DTOの取得	○	△

JPA実装を直接利用するよりも
Kuina-Daoと組み合わせて使用した方が便利！

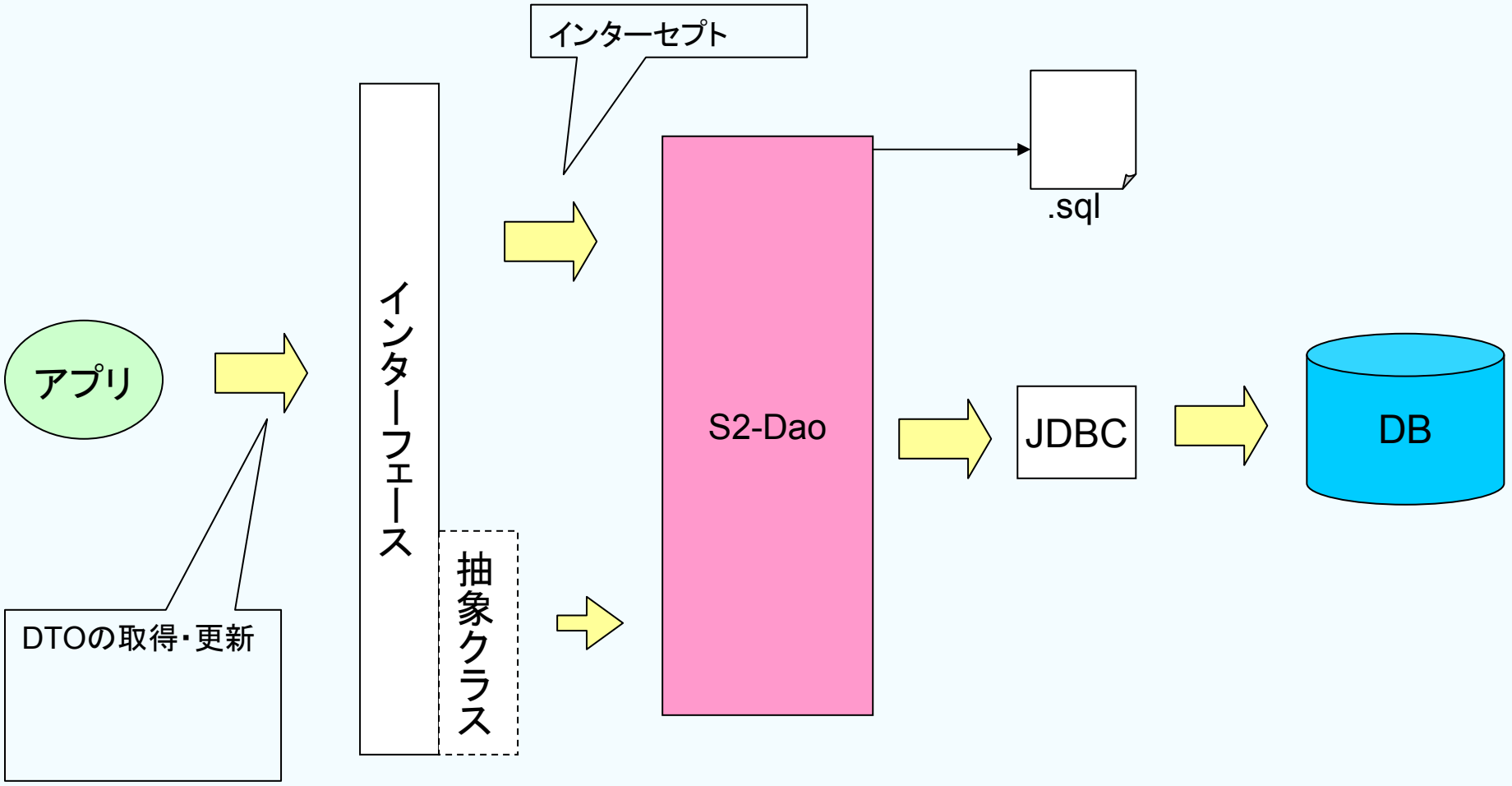
でも、そもそもJPAって実際に有用なの？

Kuina-DaoとS2Dao

Kuina-Daoの仕組み



S2Daoの仕組み



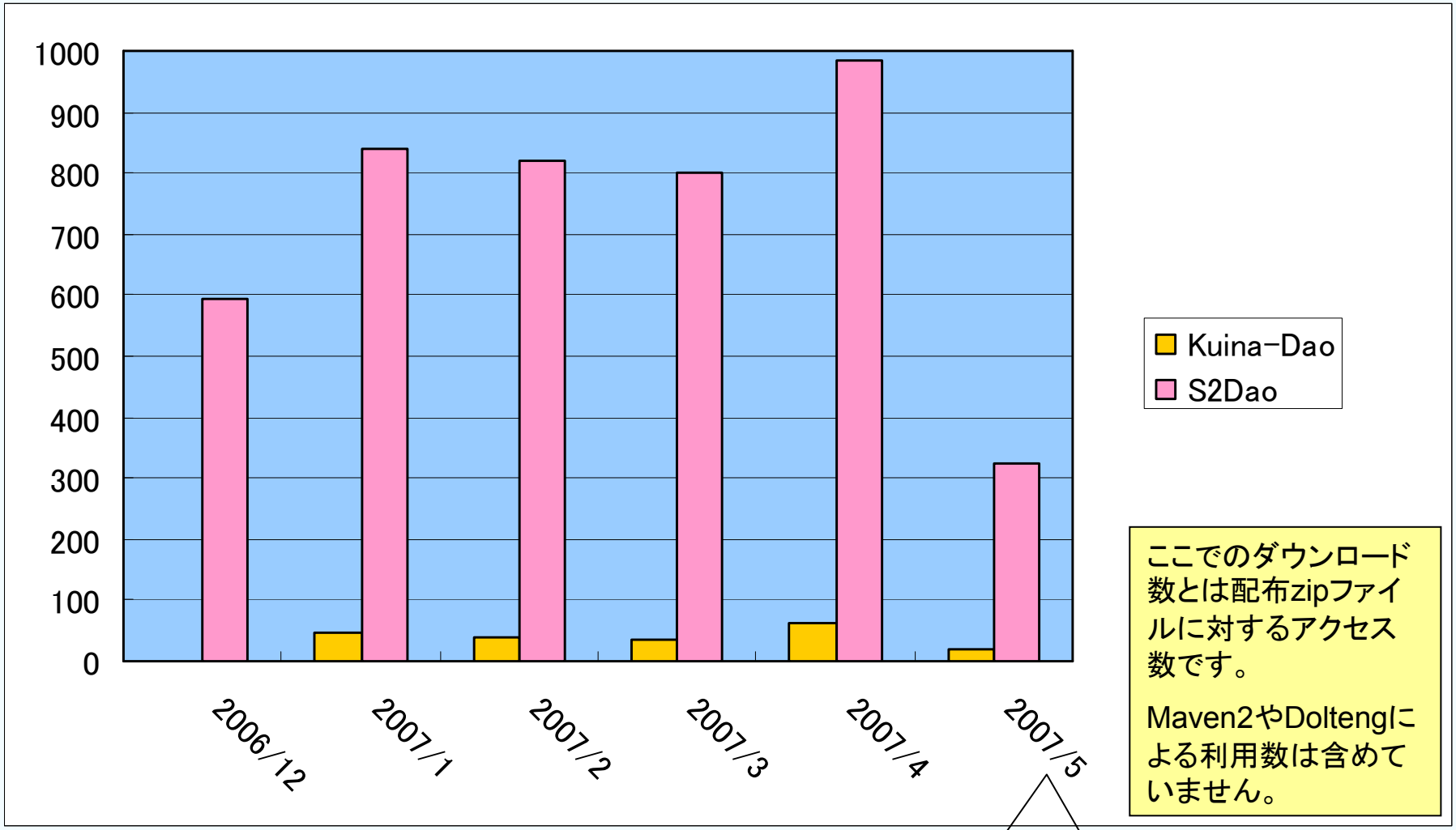
Kuina-DaoとS2Daoの関係

- ともにDaoフレームワーク
  - Kuina-DaoはJPAベース
    - 問い合わせはJPQLもしくはSQL
    - マッピングはJPAの標準仕様
    - 問い合わせの実行やマッピングはJPA実装が行う
  - S2DaoはJDBCベース
    - 問い合わせはSQL
    - マッピングはS2Daoの独自仕様
    - 問い合わせの実行もマッピングもS2Daoが行う
- アーキテクチャ上、S2Daoに対応するのはKuina-Dao単体ではなく、Kuina-Dao + JPA実装
- Kuina-DaoはS2Daoの後継ではない

(Kuina-Dao + JPA実装) VS. S2Dao

- プロジェクトへ適用する場合に、どちらがより適しているかの観点で比較

比較 1 – 人気（ダウンロード数）



比較結果 1 – 人気（ダウンロード数）

		Kuina-Dao + JPA実装	S2Dao
1	人気 （ダウンロード数）	×	○
2			
3			
4			
5			
6			
7			
8			

比較 2 – 学習コスト

- S2Daoの学習コストは低い
  - SQLの知識
  - SQLコメント
  - メソッドの命名規則
  - アノテーション

- Kuina-DAO自体の学習コストは並だが…
  - JPAの知識
    - 永続コンテキストとエンティティのライフサイクル
  - SQLの知識
  - SQLコメント
  - アノテーション
  - メソッドの命名規則
  - パラメーターの命名規約

比較結果 2 – 学習コスト

		Kuina-Dao + JPA実装	S2Dao
1	人気（ダウンロード数）	×	○
2	**学習コスト**	**×**	**○**
3			
4			
5			
6			
7			
8			

比較 3 – SQLの利用

- SQLは既存の資産
  - 実際のSQL
  - SQLの知識
  - SQLを中心とした開発に対する経験
- S2Daoの場合
  - SQLをフル活用できる
    - SQLを外部ファイルに記述しDTOにマッピングするだけ
- Kuina-Daoの場合
  - SQLをサポートするが、主役はJPAでのデータアクセス
  - 一部の複雑なアクセスはJPAを経由しないでSQLで行うのがよい
    - S2Daoと同様外部ファイルに記述したSQLをDTOにマッピングできる

比較結果 3 – SQLの利用

		Kuina-Dao + JPA実装	S2Dao
1	人気（ダウンロード数）	×	○
2	学習コスト	×	○
3	**SQLの利用**	**△**	**○**
4			
5			
6			
7			
8			

比較 4 – チューニング

- JPAを使う/使わないでチューニングの対象は異なる
- S2Daoの場合
  - SQL
- JPAの場合
  - マッピング
  - フェッチ
  - 永続コンテキスト

比較結果 4 – チューニング

		Kuina-Dao + JPA実装	S2Dao
1	人気（ダウンロード数）	×	○
2	学習コスト	×	○
3	SQLの利用	△	○
4	**チューニング**	**△**	**○**
5			
6			
7			
8			

比較 5 – レガシーなERモデルとの相性

- ここでの「レガシーなERモデル」の定義
  - 複合主キーを利用している
  - トリガーを多用している
- S2Daoの場合
  - すべてSQLで解決できるので相性は良い
- JPAの場合
  - JPAでは複合主キーを使う/使わないでマッピングが異なるので注意
    - @EmbeddedId、@JoinColumns
    - クラス数が増える、コードが増える
  - トリガーでDBが更新されると、DBとJPAの永続コンテキストがずれる
    - EntityManager#refresh()

比較結果 5 – レガシーなERモデルとの相性

		Kuina-Dao + JPA実装	S2Dao
1	人気（ダウンロード数）	×	○
2	学習コスト	×	○
3	SQLの利用	△	○
4	チューニング	△	○
5	**レガシーなERモデルとの相性**	**△**	**○**
6			
7			
8			

比較 6 – マッピング

- S2Daoの場合
  - 独自のアノテーションと規約でマッピング
    - フラットなDTOにマッピングされる
  - リレーションシップのマッピング
    - Many to One
    - 双方向関連や遅延ローディングの機能はない
- JPAの場合
  - 標準のアノテーションでマッピング
    - 階層化されたデータ構造にマッピングされる
  - リレーションシップのマッピング
    - Many to One
    - One to Many
    - One to One
    - Many to Many
    - 双方向関連や遅延ローディングをサポート

比較結果 6 – マッピング

		Kuina-Dao + JPA実装	S2Dao
1	人気（ダウンロード数）	×	○
2	学習コスト	×	○
3	SQLの利用	△	○
4	チューニング	△	○
5	レガシーなERモデルとの相性	△	○
6	**マッピング**	**○**	**△**
7			
8			

比較 7 – 永続コンテキスト

- S2Daoの場合
  - 永続コンテキストに相当するキャッシュはもたない
  - 常にDBに問い合わせることでシンプルさを保つ
- JPAの場合
  - JPA実装が永続コンテキストを管理する
  - 問い合わせはキャッシュされるので、同じトランザクション内で何度も同じエンティティにアクセス（更新、問い合わせ）する場合に効果を発揮する

比較結果 7 – 永続コンテキスト

		Kuina-Dao + JPA実装	S2Dao
1	人気（ダウンロード数）	×	○
2	学習コスト	×	○
3	SQLの利用	△	○
4	チューニング	△	○
5	レガシーなERモデルとの相性	△	○
6	マッピング	○	△
7	**永続コンテキスト**	**○**	**×**
8			

比較 8 – ツールのサポート

- S2Daoの場合
  - Seasarプロジェクトが提供
    - Dolteng
    - DBFlute
    - S2Dao-CodeGen
- Kuina-Daoの場合
  - Seasarプロジェクトが提供
    - Dolteng
  - （JPAの部分に関しては）IDEが提供
    - Eclipse
      - Dali JPA Tools
    - NetBeans
    - JDeveloper

比較結果 8 – ツールのサポート

		Kuina-Dao + JPA実装	S2Dao
1	人気（ダウンロード数）	×	○
2	学習コスト	×	○
3	SQLの利用	△	○
4	チューニング	△	○
5	レガシーなERモデルとの相性	△	○
6	マッピング	○	△
7	永続コンテキスト	○	×
8	**ツールのサポート**	**○**	**×**

Kuina-DaoとS2Dao – まとめ(1)

		Kuina-Dao + JPA実装	S2Dao
1	人気(ダウンロード数)	×	○
2	学習コスト	×	○
3	SQLの利用	△	○
4	チューニング	△	○
5	レガシーなERモデルとの相性	△	○
6	マッピング	○	△
7	永続コンテキスト	○	×
8	ツールのサポート	○	×

5 対 3 でS2Daoの勝ち？

Kuina-DaoとS2Dao – まとめ(2)

		Kuina-Dao + JPA実装	S2Dao
1	人気(ダウンロード数)	×	○
2	学習コスト	×	○
3	SQLの利用	△	○
4	チューニング	△	○
5	レガシーなERモデルとの相性	△	○
6	マッピング	○	△
7	永続コンテキスト	○	×
8	ツールのサポート	○	×

Kuina-Daoの使いどころ

2タイプのアプリケーション

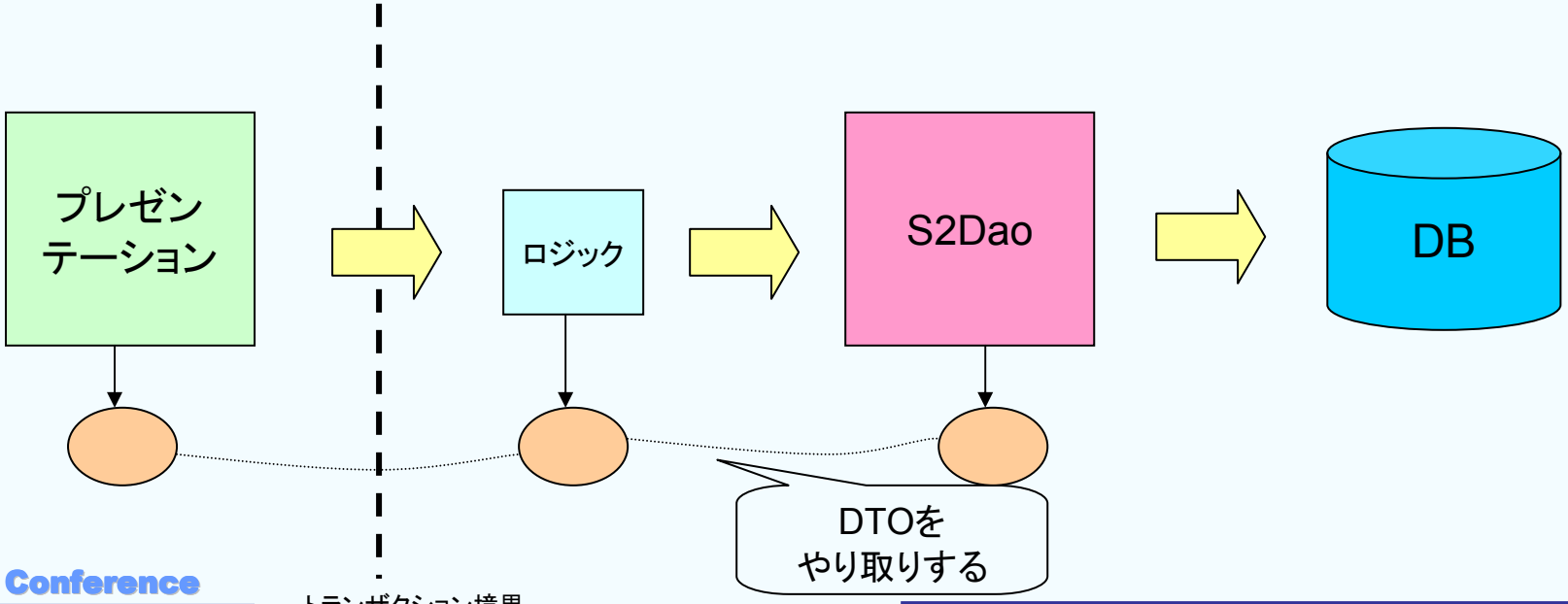
- 世の中のアプリケーションは2種類に分けられる
  - データの入出力が中心のアプリケーション
  - データの評価・加工が中心のアプリケーション

- 入出力が中心のアプリケーションとは？
  - 画面からの入力したデータをほとんどそのままバックエンドシステム（DB、メインフレーム）にわたす
  - バックエンドシステム（DB、メインフレーム）から受け取ったデータをほとんどそのまま画面に出力する

- 評価・加工が中心のアプリケーションとは？
  - データベースに問い合わせながらあるデータを評価し、その評価に基づいて加工を行うアプリケーション

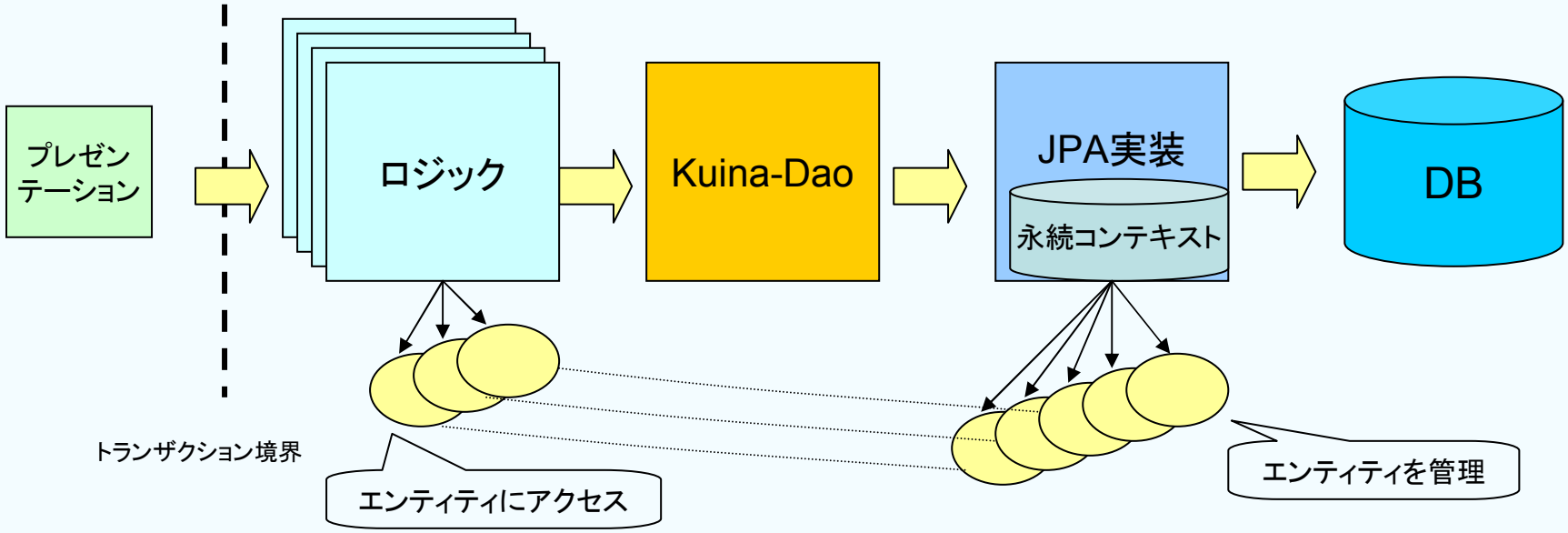
入出力中心のアプリケーションの場合

- 業務ロジックがほとんどない
  - バリデーションやデータの型変換などはある
- キャッシュの仕組みは不要
  - 入力時は、更新メソッドを呼べばよい
  - 出力時は、画面に合わせて必要なデータを取得すればよい
- 2種類のマッピングがあればOK
  - テーブルとオブジェクトのマッピング
  - 任意の結果セットとオブジェクトのマッピング



評価・加工が中心のアプリケーションの場合

- 同一テーブルの同一レコードがトランザクション内で複数のロジックにまたがって何度もアクセスされる
  - 必要毎のDBアクセスは避けたい→**永続コンテキスト**でのキャッシュ
- 同一テーブルの同一レコードがトランザクション内でさまざまに変更される
  - 変更毎のDBアクセスは避けたい→コミット時の**永続コンテキスト**のフラッシュ
- データにグループ化や階層構造が必要
  - **リレーションシップのマッピング**



評価・加工が中心のアプリケーションの例

- シミュレーション系アプリ（例えば生産管理のスケジューリング）
  - 一度のトランザクションで何度も変更されるエンティティ
    - 所要量
    - タスク
    - スケジュール
    - 在庫
  - 一度のトランザクションで何度もアクセスされるエンティティ
    - スケジュール
    - タスク
    - リソース
    - 能力
    - 部品
    - 部品構成
    - 在庫

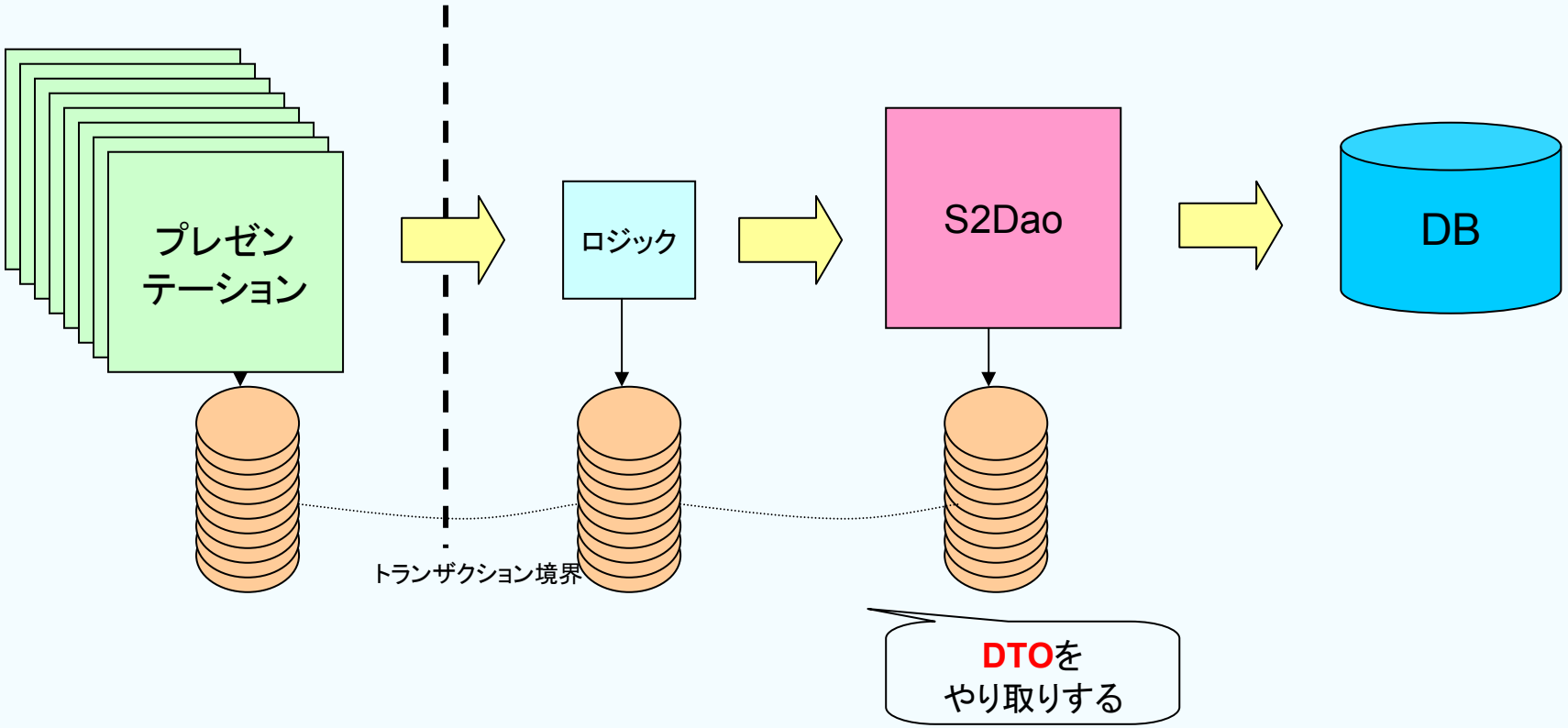
Kuina-Daoの使いどころ – まとめ

- JPAが効果的なアプリケーションにはKuina-Daoも効果的
  - Kuina-DaoはJPAを使いやすくするフレームワーク
- JPAの使いどころ
  - 永続コンテキスト
    - DBアクセスを減らすための仕組み（キャッシュ・コミット時のフラッシュ）が備わっている
      - アプリケーションをシンプルに保てる
      - 管理すべきSQLを減らせる
  - リレーションシップのマッピング
    - データのまとまりや階層構造（1対多・多対1）をロジックで扱える
  - ツール（IDEやIDEのプラグインなど）のサポート
    - エンティティの自動生成
    - エンティティの編集補完
    - 設定ファイルの自動生成

評価・加工が中心ならばKuina-Daoで
入出力中心ならばS2Daoで決まり？

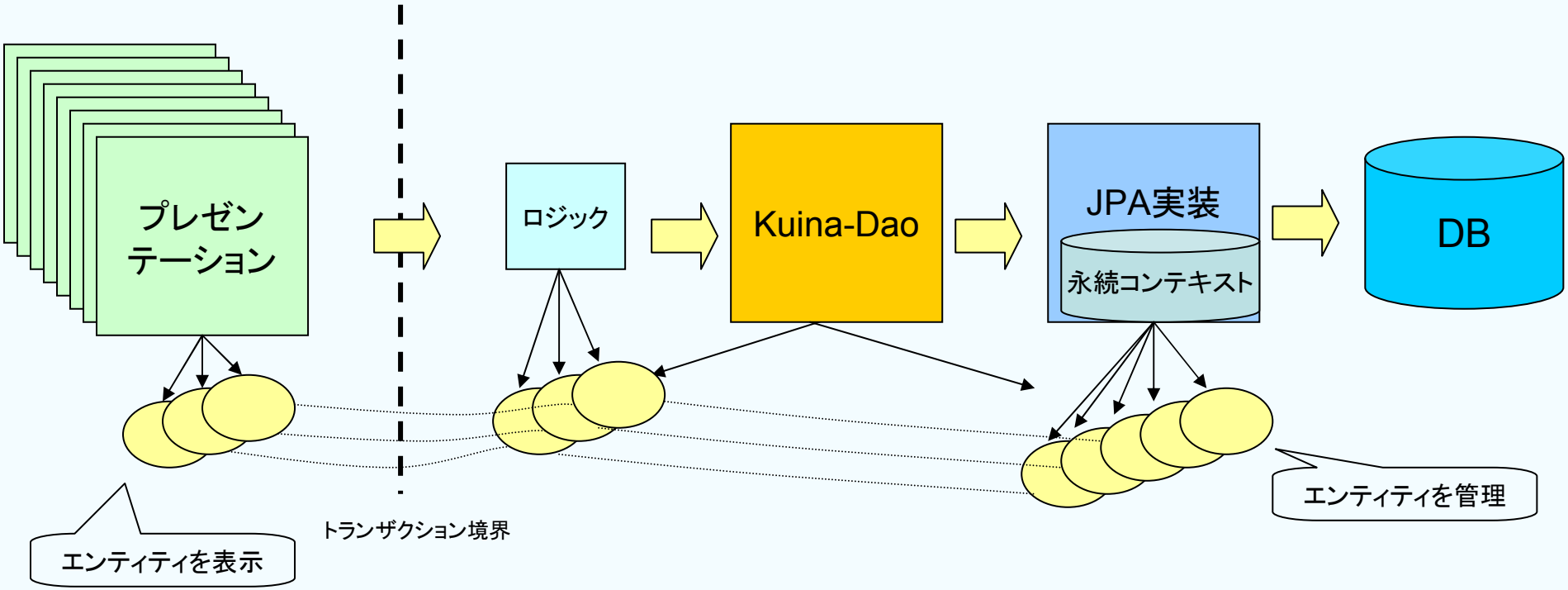
Kuina-Daoの使いどころ アドバンスド

S2Daoで入出力中心のアプリケーション



- DTOの管理が煩雑
  - 画面が増えると対応してDTOとDAOが増える

Kuina-Daoで入出力中心のアプリケーション



- エンティティをそのまま表示すればOK
  - 画面が増えてもDTOを管理する必要がない。でも…

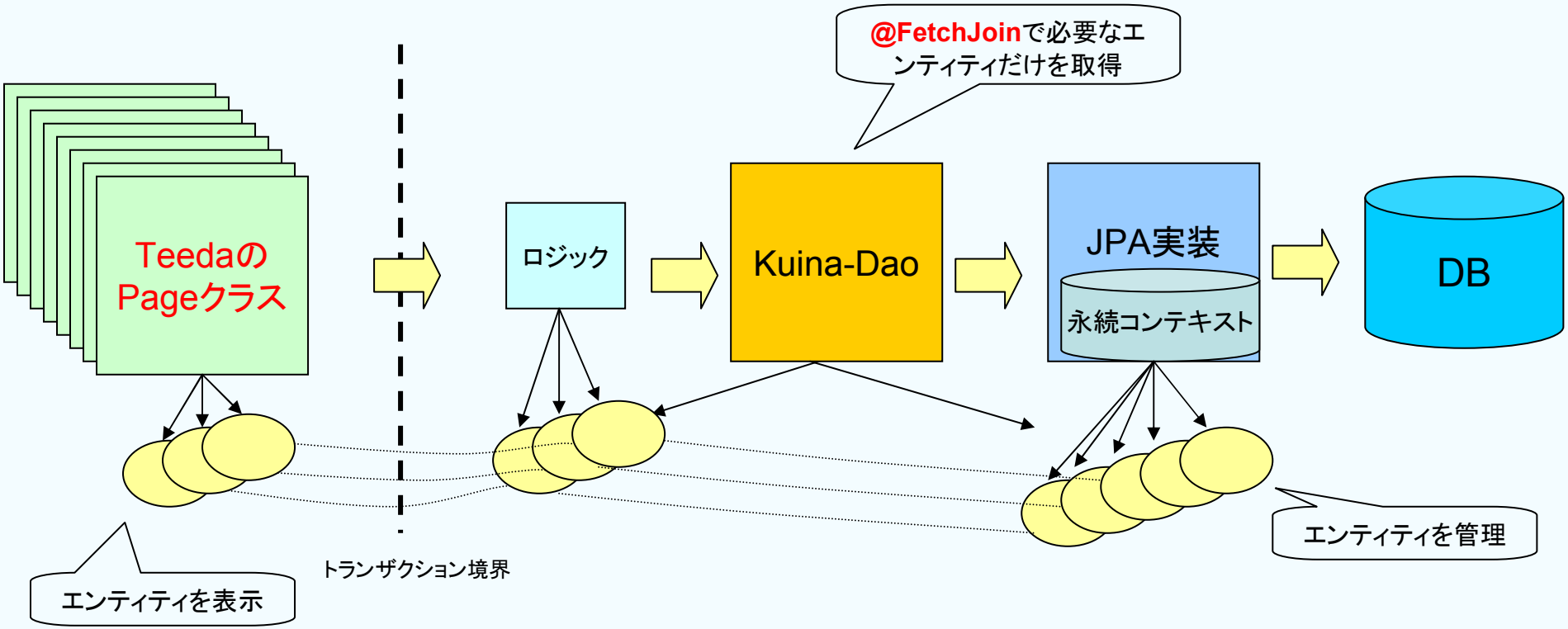
遅延ローディングにどう対応するか

a. あらかじめアクセスしておく
b. フェッチタイプをEagerに設定しておく
c. Transaction View
d. Entity Manager per Request
e. DTOに変換

遅延ローディングにどう対応するか - ベストプラクティス

- フェッチタイプはすべてLazyに設定しておく
  - @ManyToOne(**fetch = FetchType.*LAZY***)
- 必要なものだけFetch Joinで取得する
  - Kuina-Daoでは簡単にFetch Joinするための仕組みを提供
    - @Distinct
      @FetchJoin(joinSpec = JoinSpec.*LEFT_OUTER_JOIN*, value = “emps”)
      List<Dept> findByLoc(String loc);
  - もちろんJPQLを書いてもOK
    - SELECT DISTINCT dept FROM Dept AS dept LEFT OUTER JOIN FETCH dept.emps WHERE (dept.loc = :loc)

Kuina-Daoの使いどころ アドバンスド – まとめ



エンティティを直接参照する形は今後Teedaがサポート

ロードマップ

Kuina-Dao関連プロダクトのロードマップ

- 現在
  - 2007年5月 S2TopLink-JPA 1.0-r3 リリース
  - 2007年5月 S2Hibernate-JPA 1.0 リリース
  - 2007年5月 Kuina-Dao 1.0 リリース

- 予定
  - 2007年9月 S2TopLink-JPA 1.0 リリース予定
  - 2007年x月 S2OpenJPA 1.0 リリース予定
  - 2007年x月 S2Cayenne-JPA 1.0 リリース予定

皆さんの要望を受け付けています！

Kuina-Daoの関連情報

- ドキュメント
  - http://kuina.seasar.org/ja/
- 書籍
  - Java Expert #01
    - http://www.amazon.co.jp/Java-Expert-01-編集部/dp/4774130702/
- メーリングリスト
  - Seasar userメーリングリスト
    - seasar-user@ml.seasar.org
  - JPAメーリングリスト
    - jpa@ml.seasar.org
- サンプル（テストコード）
  - SVNリポジトリ
    - https://www.seasar.org/svn/kuina/trunk/kuina-dao-it
- ツール
  - Dolteng（どぅるてん）
    - http://www.seasar.org/wiki/index.php?Dolteng

ありがとうございました