Seasar Conference
2006 Autumn

Kuina-DaoとDoltengによる Easy Enterprise

2006.11.12

小林浩一 (koichik)

自己紹介

- 名前：小林 浩一
- ハンドル：koichik
- 日記：【見覚え】koichikのひとりごと【あります】
  - http://d.hatena.ne.jp/koichik/
- コミッタ
  - Seasar2，Kuina，S2Hibernate，S2TopLink
  - S2Remoting，S2RMI，S2Axis，S2JMS，S2JCA
  - S2JSF，Teeda
  - Chura，Kijimuna

アジェンダ

- JPA
  - JPAとは何か？
  - JPAの使い方
- Kuina-Dao
  - Kuina-Daoとは何か？
  - Kuina-Daoの使い方
  - ロードマップ
- Dolteng

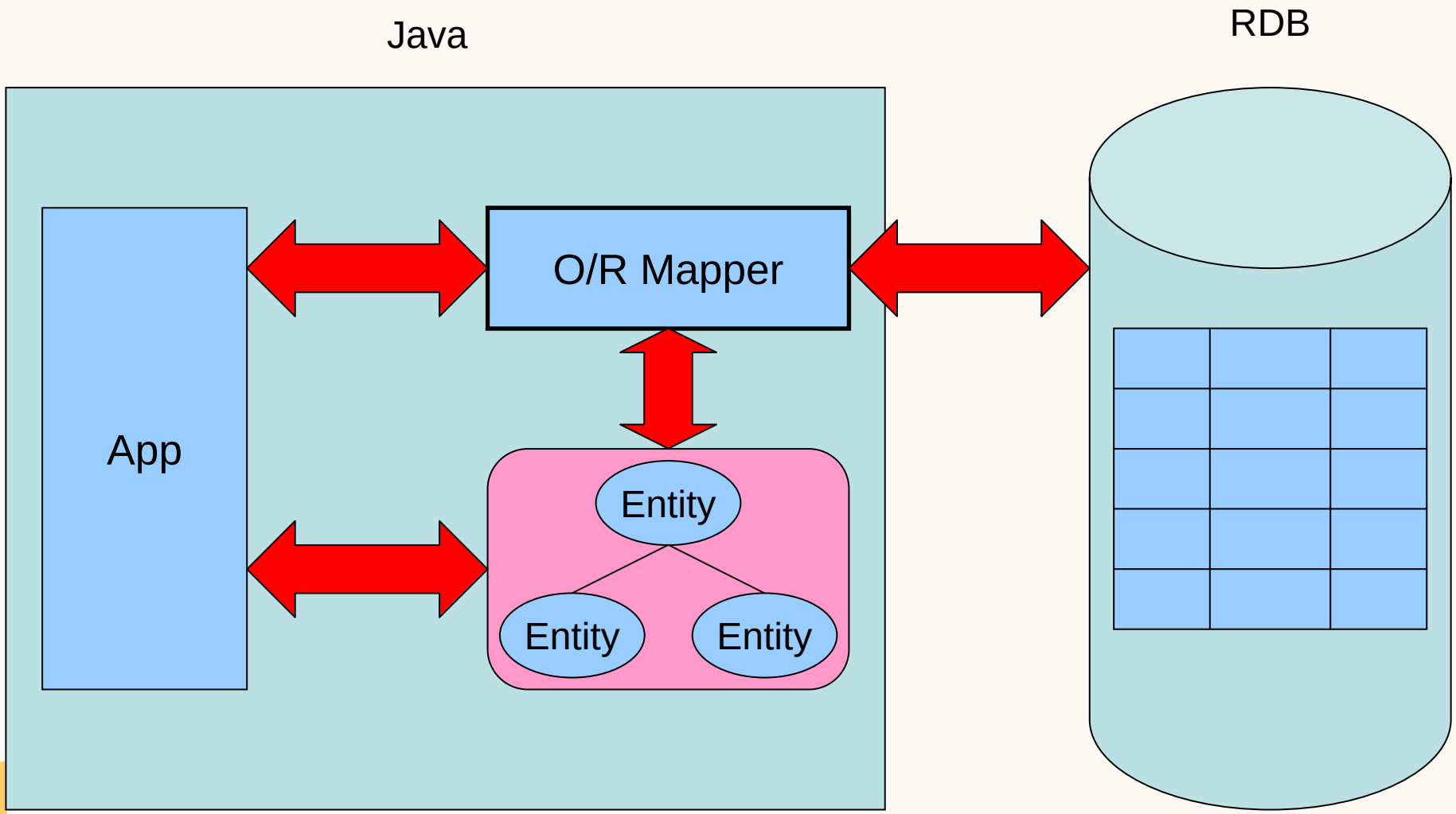
JPAとは何か？

- Java Persistence API
- 2006/05/11リリース
- Java Enterprise Edition 5.0の一部
  - Java Statndard Edition 5.0でも利用可
- Enterprise JavaBeans3.0 (JSR-220) の一部
  - EJB2.1以前のEntityBeanを代替
  - EJBコンテナなしでも利用可
- Object/Relational MappingのJava標準
  - Hibernate，Oracle TopLinkの影響大

O/R Mapping

- Javaのクラス(オブジェクト)とRDBのテーブル(レコード)をマッピング
  - マッピングはアノテーションで指定
  - XMLでも指定可
- アプリケーションは主にJavaクラスを扱う
  - 検索にはJPQLまたはSQLを使う
- RDBへの操作はO/R Mapperが行う
  - SQLの組み立てと発行
  - 結果セットからオブジェクトの生成と値の反映

O/R Mapper



JPAの使い方

- 主な要素
  - 永続コンテキスト
  - EntityManager
  - JPQL
- EntityManagerとDAO

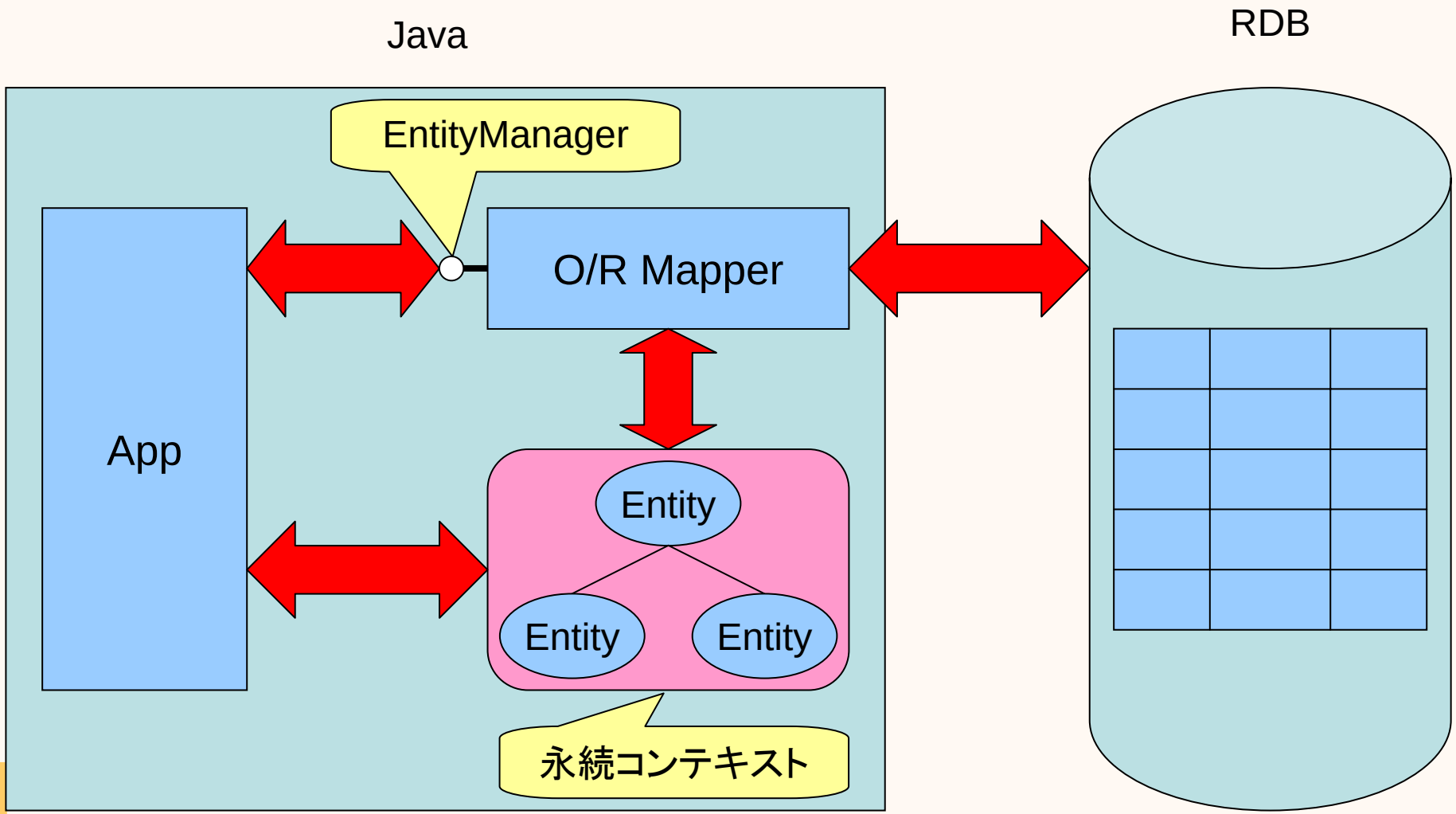
永続コンテキスト

- O/R Mapperが管理するエンティティオブジェクトの集まり
  - エンティティオブジェクトのキャッシュ
- RDBに対する操作 == 永続コンテキストの操作
  - SELECT：永続コンテキストにエンティティをロードする
  - INSERT：永続コンテキストにエンティティを追加する
  - UPDATE：永続コンテキスのエンティティを変更する
  - DELETE：永続コンテキストからエンティティを削除する
- トランザクションに関連づけられる
  - トランザクションコミット時にエンティティとRDBのデータを同期
    - INSERT/UPDATE/DELETEが実行
  - ロールバック時には同期しない
- 2つのタイプ
  - トランザクションスコープ永続コンテキスト
  - 拡張永続コンテキスト (SFSBでのみ利用可)

EntityManager

- javax.persistence.EntityManager
  - 永続コンテキストを操作するための Javaインタフェース
  - O/R Mapperが実装を提供
  - アプリケーションから利用する

永続コンテキストとEntityManager



EntityManagerの主なメソッド

- 照会系
  - find(), getReference()
  - createNamedQuery(), createNativeQuery(), createQuery()
- 更新系
  - persist(), remove()
- その他
  - contains(), lock(), merge(), refresh()
- 永続コンテキストの操作
  - clear(), close(), flush(), setFlushMode()

JPQL

- Java Persistence Query Language
- JPAの問い合わせ言語
  - SQLの貧弱な方言(サブセット)
    - SELECT
    - UPDATE
    - DELETE
  - 結合の指定がSQLよりちょっと楽
  - 高度なことはできない
    - 副問い合わせはWHERE句のみ
    - 関数・式が貧弱
    - 将来の強化に期待

EntityManagerとDAO

- EntityManagerがあればDAOは不要？
  - EntityManagerは汎用のDAOみたいなもの？
  - わざわざDAO作るのは面倒？

EntityManager == DAO？

- 主キーで検索
  - EntityManager

```
private EntityManager em;
…
Employee emp = em.find(Employee.class, 100);
```

  - DAO

```
private EmployeeDao dao;
…
Employee emp = dao.find(100);
```

EntityManager == DAO？

- 永続化
  - EntityManager

```
private EntityManager em;
…
em.persist(emp);
```

  - DAO

```
private EmployeeDao dao;
…
dao.persist(emp);
```

EntityManager == DAO？

- JPQLで問い合わせ
  - EntityManager

```
private EntityManager em;
・・・
List<Employee> emps = (List<Employee>)
    em.createQuery("SELECT e FROM Employee e WHERE "
            + "e.name=:name AND e.dept=:dept")
        .setParameter("name", name)
        .setParameter("dept", dept)
        .getResultList();
```

EntityManager == DAO？

- 動的な問い合わせ
  - EntityManager

```
StringBuilder queryString = new StringBuilder(
    "SELECT e FROM Employee e ");
if (name != null) {
    buf.append("WHERE e.name=:name");
}
Query query = em.createQuery(new String(queryString));
if (name != null) {
    query.setParameter("name", name);
}
List<Employee> emps = (List<Employee>) query.getResultList();
```

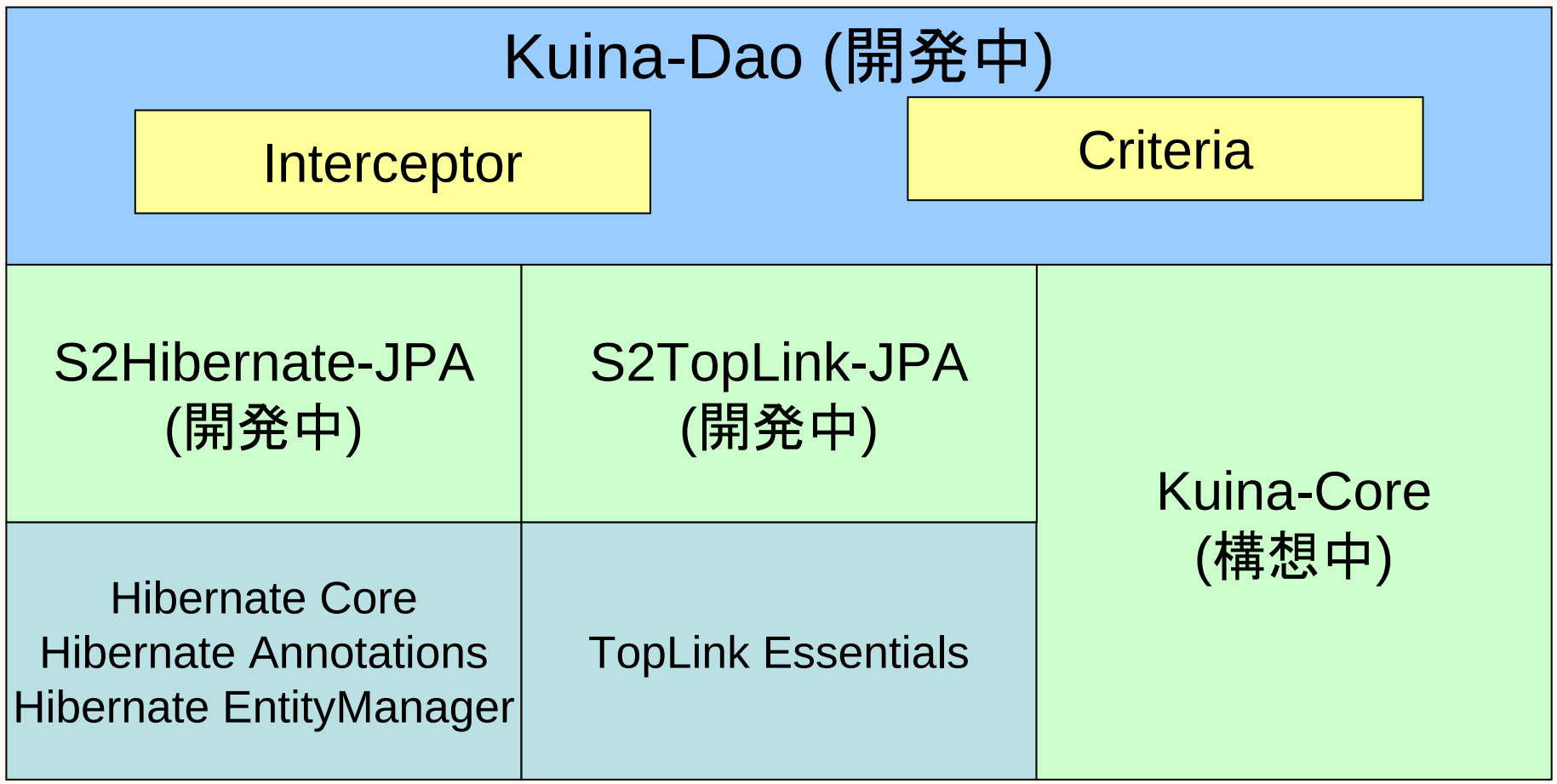
EntityManager != DAO！

- 問い合わせのコードが煩雑になりやすい
  - 動的な問い合わせ文字列の作成
  - パラメータの設定
- JPAでもDAOは必要！
  - Kuina-Dao
- DAOを作るのが面倒？
  - Dolteng

Kuina-Daoとは？

- Kuina
  - JPA関連プロダクトを開発するプロジェクト
  - http://kuina.seasar.org/ (工事中)
- Kuina-Core
  - JPAの実装
  - 着手は来年？
- Kuina-Dao
  - JPA上のDAOフレームワーク
  - 絶賛開発中

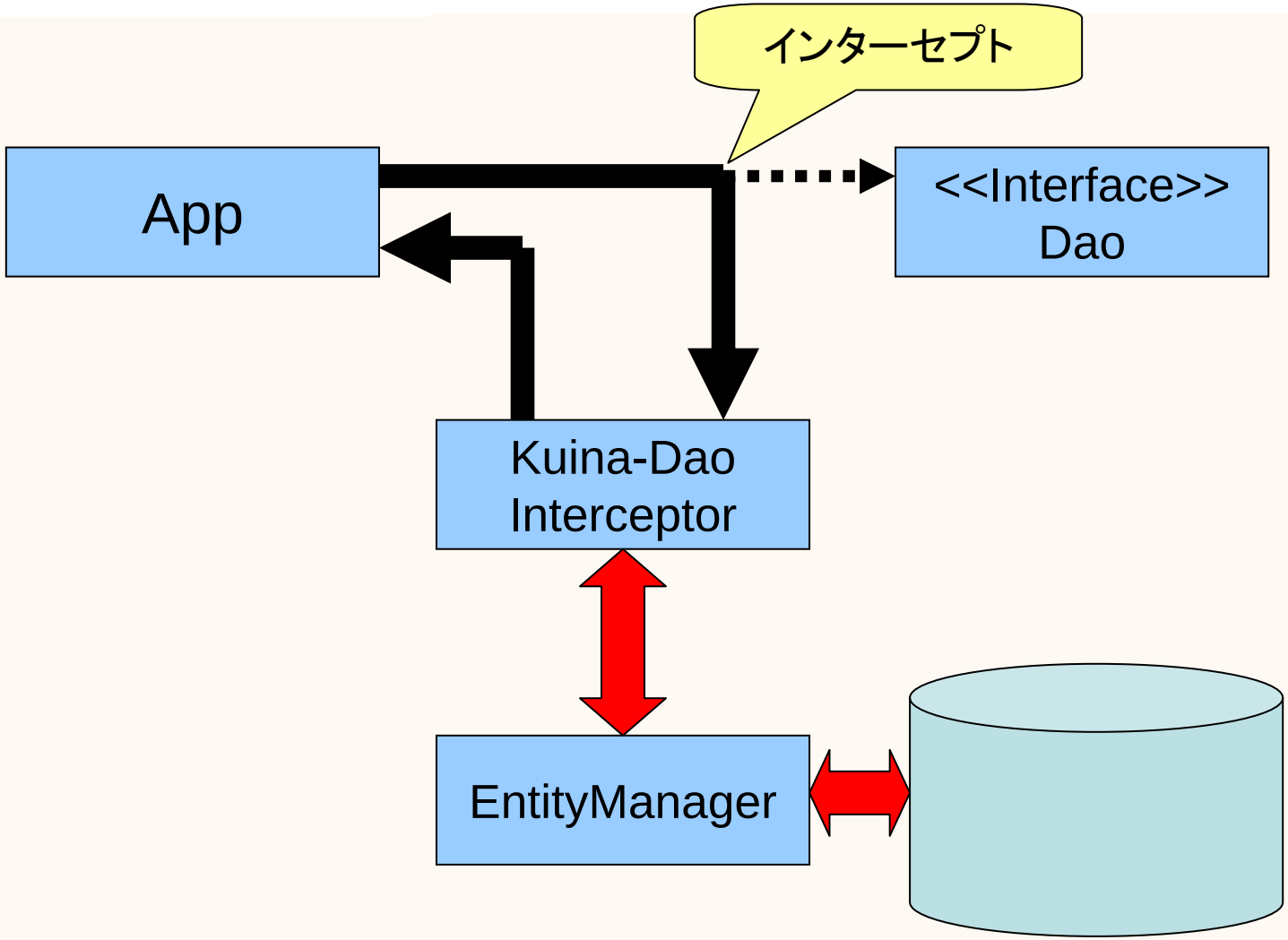
Kuina-Daoの位置づけ



Kuina-Daoの構成

- Kuina-Dao Interceptor
  - JPA版S2Dao
  - DAOの実装をAOPで提供
  - 利用者はJavaインタフェースを定義するだけ
- Criteria API
  - JPQLを組み立てるためのAPI
  - Dao実装クラスから利用
  - Kuina-Dao Interceptor内部でも利用している
  - 今回は割愛

Kuina-Dao Interceptor



Kuina-Daoの使い方

- Daoインタフェースを用意する
  - 単なるJavaインタフェース
  - パッケージとクラス名
    - Seasar2.4の規約に従う
    - <ルートパッケージ>.dao.<エンティティ名>Dao
  - メソッド名
    - Kuina-Daoの規約に従う
  - 引数名
    - 問い合わせ条件で使用
    - Diiguで引数名をクラスファイルに埋め込む

DAOのメソッド

- 照会系
  - find(), getReference()
  - findAll()
  - **find～(), get～()**
- 更新系
  - persist(), remove()
- その他
  - contains(),
  - merge(), refresh()
  - readLock(), writeLock()

DAOインタフェースの例

```
public interface EmployeeDao {
    Employee find(Integer id);
    List<Employee> findAll();
    List<Employee> findByName(String name);
    List<Employee> findByExample(Employee example);
    List<Employee> findByDto(EmployeeDto dto);
    void persist(Employee emp);
    void remove(Employee emp);
    boolean contains(Employee emp);
    Employee merge(Employee emp);
    void refresh(Employee emp);
    void readLock(Employee emp);
    void writeLock(Employee emp);
    ...
}
```

Kuina-Daoの問い合わせ機能

- Named Query
  - JPAの名前付き問い合わせ(静的)
- Query by Examle
  - エンティティのプロパティを条件とする問い合わせ(動的)
- Query by DTO
  - DTOのプロパティを条件とする問い合わせ(動的)
- Query by Parameter
  - メソッドの引数を条件とする問い合わせ(動的)
- SQL Query
  - S2Dao/Uujiと同等の2Way SQLで問い合わせ(動的)

Named Query

- EntityManager#createNamedQuery()を呼び出す
- メソッドの名前
  - find～()
  - get～()
- 戻り値の型
  - find～()はエンティティのList
  - get～()はエンティティ
- 引数
  - Named Query中のパラメータと一致
- Named Queryの名前
  - <エンティティ名>.<メソッド名>

NamedQueryの例

ORMファイル (EmployeeOrm.xml)

```
<named-query name="Employee.findByName">
    <query><![CDATA[
    SELECT emp FROM Employee emp WHERE emp.name = :name
    ]]></query>
</named-query>
```

```
public interface EmployeeDao {
    List<Employee> findByName(String name);
...
}
```

Named Queryの名前は
Employee.findByName

パラメータの名前を
一致させる

Query by Example

- EntityManager#createQuery()を呼び出す
- メソッドの名前
  - find～()
  - get～()
- 戻り値の型
  - find～()はエンティティのList
  - get～()はエンティティ
- 第1引数はエンティティ
  - エンティティのnullでないプロパティが問い合わせ条件に含められる
  - 問い合わせ条件の演算子は = のみ

Query by Exampleの例

```
public interface EmployeeDao {
    List<Employee> findByExample(Employee emp);
}

public class EmployeeLogicImpl {
    public void foo() {
        ...
        Employee ex = new Employee();
        ex.setName(name);
        ex.setBirthday(birthday);
        List<Employee> emp = dao.findByExample(ex);
        ...
    }
}
```

```
SELECT emp
FROM   Employee emp
WHERE  emp.name = :name
    AND  emp.birthday = :birthday
```

Query by DTO

- EntityManager#createQuery()を呼び出す
- メソッド名・戻り値型はQuery by Exampleと同じ
- 第1引数はDTO
  - DTOのnullでないプロパティが問い合わせ条件に含められる
- DTOのプロパティ名
  - エンティティのプロパティ名 [+ '_' + 演算子]
    - EQ(=), NE(<>),
    - GT(>), GE(>=), LT(<), LE(<=)
    - LIKE, STARTS, ENDS, CONTAINS
    - IN, IS_NULL, IS_NOT_NULL
    - 演算子の指定を省略した場合はEQ
  - '$'区切りで関連プロパティのプロパティを指定可

Query by DTOの例

```
public interface EmployeeDao {
    List<Employee> findByDto(EmployeeDto dto);
}
public class EmployeeDto {
    private String Name_LIKE;
    private String birthday _GT;
}
public class EmployeeLogicImpl {
    public void foo() {
        EmployeeDto dto = new EmployeeDto();
        dto.setName_LIKE(pattern);
        dto.setBirthday_GT(aDay);
        List<Employee> emp = dao.findByDto(dto);
  }
}
```

```
SELECT emp FROM Employee emp
WHERE emp.name LIKE :name AND emp.birthday > :birthday
```

Query by Parameter

- EntityManager#createQuery()を呼び出す
- メソッド名・戻り値型はQuery by Exampleと同じ
- 引数の数は任意
  - nullでない引数が問い合わせ条件に含められる
- 引数の名前
  - エンティティのプロパティ名 [+ '_' + 演算子]
    - EQ(=), NE(<>),
    - GT(>), GE(>=), LT(<), LE(<=)
    - LIKE, STARTS, ENDS, CONTAINS
    - IN, IS_NULL, IS_NOT_NULL
    - 演算子の指定を省略した場合はEQ
  - '$'区切りで関連プロパティのプロパティを指定可

Query by Parameterの例

```
public interface EmployeeDao {
    List<Employee>
      findByDepartmentName(String department$name_LIKE);
    ...
}

public class EmployeeLogic {
    public void foo() {
        ...
        Employee emp = dao.findByDepartmentName(deptName);
        ...
    }
}
```

departmentはEmployeeの関連プロパティ

nameはDepartmentのプロパティ

```
SELECT emp
FROM Employee emp INNER JOIN department dept
WHERE dept.name LIKE :department$name
```

SQL Query

- EntityManagerを呼び出さない
  - JDBCを直接呼び出す
- メソッド名はQuery by Exampleと同じ
- 戻り値の型
  - find～()はDTOのList
  - get～()はDTO
- 引数の数は任意
  - SQLファイル中のバインド変数コメントと一致
- SQLファイル
  - <DAOクラス名>_<メソッド名>.sql
  - Uuji，S2Daoと同等

SQL Queryの例

SQLファイル (EmployeeDao_findByName.sql)

```
SELECT id, name FROM Employee WHERE name = /*name*/'Foo'
```

```
public interface EmployeeDao {
    List<EmpDto> findByName(String name);
...
}
```

.sqlファイルの名前は
EmployeeDao_findByName.sql

パラメータの名前を
一致させる

その他

- ORDER BY (Named Query，SQL Queryを除く)
  - 引数またはDTOのプロパティで指定可
  - 引数名またはプロパティ名
    - orderBy
  - 型
    - String，String[]，OrderbySpec，OrderbySpec[]
- ページング (SQL Queryを除く)
  - 引数またはDTOのプロパティで指定可
  - 引数名またはプロパティ名
    - firstResult，maxResults
  - 型
    - int

問い合わせの使い分け

- エンティティを取得
  - 静的な問い合わせ
    - NamedQuery
  - 動的な問い合わせ
    - 問い合わせに含める条件が少ない
      - Query by Parameter
    - 問い合わせ条件が=のみ・Pageクラスと連携
      - Query by Example
    - その他
      - Query by DTO
- 任意の結果セットを取得
  - SQL Query

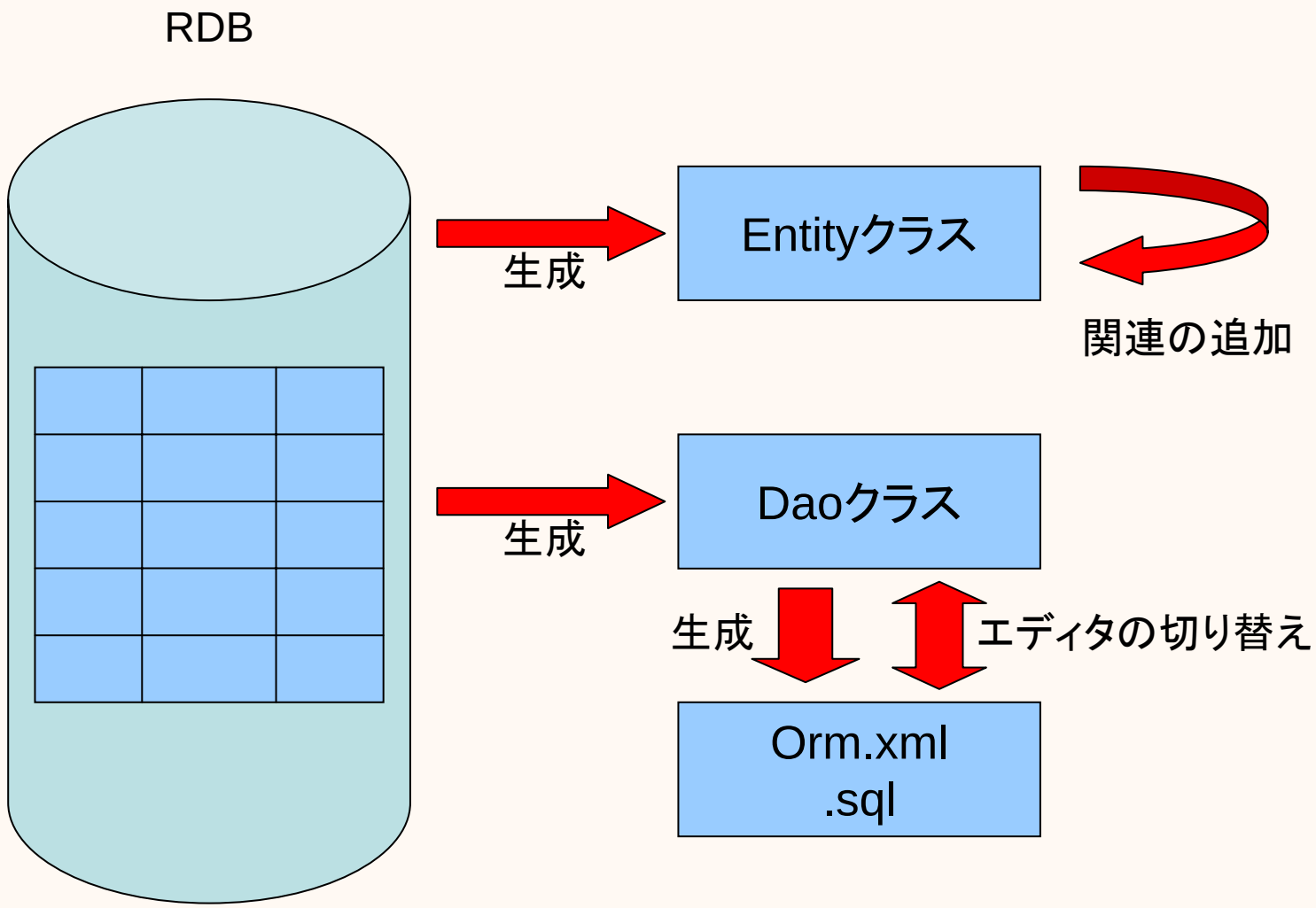
ロードマップ

- Kuina-Dao 1.0.0リリースに向けて
  - ドキュメント
  - ドキュメント(笑)
  - ドキュメント(苦笑)
  - 機能強化
    - FETCH JOINのサポート
    - 動的なバルクUPDATE・DELETE
      - Named QueryによるバルクUPDATE・DELETEは対応済み
    - 集計関数(COUNT，MAX，MIN，AVG)のサポート
    - 2Way JPQL

Dolteng

- Churaプロジェクト
  - http://chura.seasar.org/
  - Seasar2.4を中心としたAll In Oneパッケージを提供するプロジェクト
- Dolteng
  - Churaを実現するEclipseプラグイン
  - 更新サイト
    - http://eclipse.seasar.org/updates/3.1beta/
  - テーブル定義からエンティティ・DAOのソースファイル生成
  - DAOソースファイルとOrm.xmlまたは.sqlファイル間のエディタ切り替え

DoltengのJPAサポート



ご清聴

ありがとうございました