JPA & Kuina-Dao入門

2007.11.11

The Seasar Project
中村年宏(taedium)

自己紹介

- 中村年宏
  - ブログ
    - http://d.hatena.ne.jp/taedium/
  - メールアドレス
    - toshihiro.nakamura@gmail.com
  - コミッタとして関わっているプロダクト
    - S2Container、S2Dao、S2Hibernate、Kuina-Dao など
  - 執筆活動
    - 記事：EJB3.0入門講座
      http://itpro.nikkeibp.co.jp/article/COLUMN/20060615/241006/?ST=develop
    - 書籍：JPA入門（絶賛執筆中）
      来年初頭発売？

アジェンダ

- JPAとは？
- JPAの効率的な学習方法
- Kuina-Daoとは？
- Q&A

アジェンダ

- JPAとは？
- JPAの効率的な学習方法
- Kuina-Daoとは？

JPAとは？

- **J**ava **P**ersistence **A**PI
  - Javaの永続化とO/Rマッピングの標準API
    - 要するにデータベースアクセスのAPI
  - JavaEE環境でもJavaSE環境でも使用できる
    - ただし、JavaSE5以上が必須
  - EJB3.0とは別もの
  - JPAはあくまで仕様

代表的なJPA実装プロダクト

- Hibernate
  - Red Hat社が提供のOSS。いわずと知れたO/Rマッピングフレームワークの代名詞。
- TopLink Essentials
  - The GlassFish communityが提供のOSS。Oracle社のTopLinkがベースになっている。
- Apache OpenJPA
  - The Apache Software Foundationが提供のOSS。BEA社のKodoがベースになっている。

JPAはJDBCとどう違う？

- O/Rマッピングの自動化
- SQLの方言の吸収
  - ページングのSQLなど
- キャッシュ機能

JPAはJDBCとどう違う？

- 部署テーブルを使って比較してみる。

DEPT（部署）

PK

ID	DEPT_NO（部署番号）	DEPT_NAME（部署名）	LOC（所在地）	VERSION_NO（バージョン番号）
1	10	ACCOUNTING	NEW YORK	0
2	20	RESERCH	DALLAS	0
3	30	SALES	CHICAGO	0
4	40	OPERATIONS	BOSTON	0

O/Rマッピング Javaクラスのちがい

通常のJavaBeans

JDBCの場合

```
public class Dept {
  private Long id;
  private Integer deptNo;
  private String deptName;
  private String loc;
  private Integer versionNo;

  public Long getId(){...}
  public void setId(Long id){...}
  ...
}
```

エンティティクラス

JPAの場合

```
@Entity
public class Dept {

  @Id
  @GeneratedValue
  private Long id;

  @Column(name = "DEPT_NO")
  private Integer deptNo;

  @Column(name = "DEPT_NAME")
  private String deptName;

  private String loc;

  @Version
  @Column(name = "VERSION_NO")
  private Integer versionNo;


  public Long getId(){...}

  public void setId(Long id){...}

  ...
}
```

アノテーションを指定する。

O/Rマッピング
主キーで取得するケース

JDBCの場合

```
public Dept getDept(long id) throws SQLException {
  String sql =
    "select ID, DEPT_NO, DEPT_NAME, LOC, VERSION_NO from DEPT where ID = ?";
  Connection con = dataSource.getConnection();
  PreparedStatement ps = con.prepareStatement(sql);
  ps.setLong(1, id);
  ResultSet rs = ps.executeQuery();
  if (rs.next()) {
    Dept dept = new Dept();
    dept.setId(rs.getLong(1));
    dept.setDeptNo(rs.getInt(2));
    dept.setDeptName(rs.getString(3));
    dept.setLoc(rs.getString(4));
    dept.setVersionNo(rs.getInt(5));
    return dept;
  }
  return null;
}
```

コード上で
マッピングを行う。

リソースの開放処理は省略しています。

O/Rマッピング
主キーで取得するケース

JPAの場合

```
public Dept getDept(long id) {
  return entityManager.find(Dept.class, id);
}
```

すでに
アノテーションで
マッピングが行われている
ためコードが簡潔で済む。

O/Rマッピング
新規追加するケース

JDBCの場合

```
public void insert(Dept dept) throws SQLException {
  String sql =
    "insert into DEPT (ID, DEPT_NO, DEPT_NAME, LOC, VERSION_NO) " +
    " values (?, ?, ?, ?, ?)";
  Connection con = dataSource.getConnection();
  PreparedStatement ps = con.prepareStatement(sql);
  ps.setLong(1, dept.getId());
  ps.setInt(2, dept.getDeptNo());
  ps.setString(3, dept.getDeptName());
  ps.setString(4, dept.getLoc());
  ps.setInt(5, dept.getVersionNo());
  ps.executeUpdate();
}
```

コード上で
マッピングを行う。

リソースの開放処理は省略しています。

O/Rマッピング
新規追加するケース

JPAの場合

```
public void insert(Dept dept) {
  entityManager.persist(dept);
}
```

すでに
アノテーションで
マッピングが行われている
ためコードが簡潔で済む。

ページング

RDBMSごとにSQLを別のものにする必要がある。

JDBCでOracleを使う場合

```
select * from ( select temp_.*, rownum rownum_ from (
  select T.dept_id, T.dept_no, T.dept_name, T.loc,
  T.version FROM DEPT T order by T.dept_name ) temp_
  where rownum <= ? ) where rownum_ > ?
```

JDBCでH2を使う場合

```
select T.dept_id, T.dept_no, T.dept_name, T.loc,
  T.version FROM DEPT T order by T.dept_name limit ?
  offset ?
```

ページング

どのRDBMSであっても同じ呼び出し方でOK。

JPAの場合

```
public List<Dept> getDeptList(int offset, int limit) {
  return entityManager
    .createQuery("select d from Dept d order by d.deptName")
    .setFirstResult(offset)
    .setMaxResults(limit)
    .getResultList();
}
```

キャッシュ機能

JDBCの場合

キャッシュの機能はない。

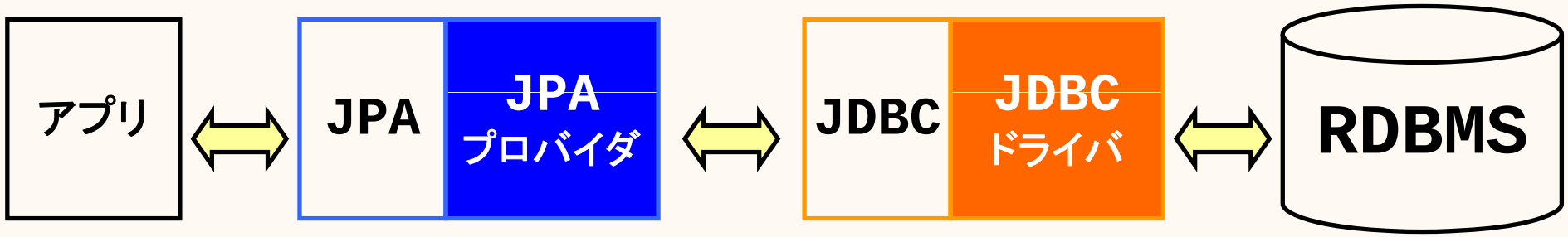
JPAの場合

```
entityManager.find(Dept.class, 1L);
entityManager.find(Dept.class, 1L);
entityManager.find(Dept.class, 1L);
```

実際のデータベースアクセスは最初の一回だけ。

JDBCとJPAの関係

- JPAはアプリケーションにとってJDBCよりも使いやすいAPI。
- JPAプロバイダ（HibernateやOpenJPA）は内部的にJDBCを使ってRDBMSにアクセスする。



アジェンダ

- JPAとは？
- JPAの効率的な学習方法
- Kuina-Daoとは？

学習のポイント

- 多くの機能に惑わされない
  - よく使うのは一握り、最初に代表的なアノテーションを覚えてしまうのがいい
- 実際に動かしてみる
  - 環境を準備するのに時間がかかると嫌になってしまう、便利な開発環境を利用するのがいい

最初はこれだけ覚えれば大丈夫！ JPAのアノテーション

- @Entity
- @Table
- @Colum
- @Id
- @GeneratedValue

```
@Entity
@Table(name=“DEPT”)
public class Department {
  @Id
  @GeneratedValue
  private Long id;
  @Column(name = "DEPT_NO")
  private Integer deptNo;
  @Column(name = "DEPT_NAME")
  private String deptName;
  private String loc;
  @Version
  @Column(name = "VERSION_NO")
  private Integer versionNo;

  ...
}
```

次はこれだけ覚えれば大丈夫！！ JPAのアノテーション

- @OneToMany
- @ManyToOne

```
@Entity
public class Dept {
 @Id
 @GeneratedValue
 private Long id;
 @OneToMany(mappedBy = "dept")
 private Set<Emp> emps = new HashSet<Emp>();
 ...
}
```

```
@Entity
public class Emp {
@Id
@GeneratedValue
private Long id;
@ManyToOne(fetch = FetchType.LAZY)
private Dept dept;
  ...
}
```

必ずLAZYを指定。

@ManyToManyや@OneToOneというアノテーションもあるが、ちゃんと理解できるまでは使わないほうが安全。

存在は確認しておきたい JPAの設定ファイル persitence.xml

- META-INFの直下に必須
  - 次のことを設定できる
    - 永続プロバイダ
    - データソース
    - トランザクションのタイプ
    - 接続先のRDBMSの種類

JPAを試すのにお奨めの開発環境

- Hibernate
  - エンティティクラスからテーブルを自動生成できる
- Seasar2
  - テストツール（S2UnitまたはS2JUnit4）が便利
  - トランザクションやデータソースなどHibernateとの連携をサポート
  - エンティティクラスの自動検出ができる
- Dolteng（どぅるてん）
  - Seasar2 + Hibernateに必要なファイル群を自動生成できる
  - エンティティクラスを自動生成できる
  - H2のサンプルデータをもつ

デモ

- デモ1
  - 開発環境の作成（Dolteng 0.23.0を使用します）
- デモ2
  - テーブルからエンティティクラスを作って動かす
- デモ3
  - エンティティクラスからテーブルを作って動かす
- デモ4
  - 関連を作成する

JPAを使いこなすためのポイント

- 複合主キーは使わない
  - 主キーはサロゲートキー1つとする
- エンティティクラスとテーブルは1:1とする
  - ちゃんと理解するまでは継承や組み込みオブジェクトは使わない
- すべてをJPAで解決しようとしない
  - 必要ならばSQLも使用する
- エンティティにロジックをもたせない
  - 真のオブジェクト指向とかを求めない
- FetchType.LAZYとFetch Joinを効果的に利用する
  - 必要なデータにだけアクセスする

アジェンダ

- JPAとは？
- JPAの効率的な学習方法
- Kuina-Daoとは？

Kuina-Daoとは？

- JPA上で利用可能なDaoフレームワーク
  - http://kuina.seasar.org/ja/
  - Daoインタフェースさえ定義すれば実装がいらない
- 複数のJPA実装に対応
  - Hibernate、TopLink Essentials、OpenJPA
- メソッドの引数名を実行時に利用する
- JPAの使いづらいところを解決！
  - 動的なクエリの自動生成
  - SQLのDTOへのマッピング

Daoインタフェースのみで動作

エンティティクラスDeptに対するDao

```
public interface DeptDao {
  /** 全件取得 */
  public List<Dept> findAll();
  /** IDで取得 */
  public Dept find(Long id);
  /** 追加 */
  public void persist(Dept dept);
  /** 削除 */
  public void remove(Dept dept);
}
```

メソッドの引数名を実行時に利用

- Diiguを利用
  - Java クラスまたはインタフェースのメソッドが持つ引数の名前を実行時に利用可能にするためのプロダクト。

```
public interface DeptDao {
    public List<Dept> findDeptByDeptName(String deptName);
}
```

DiiguによりKuina-DaoはdeptNameをwhere句に含めるべきと判断できる。そして、下記のJPAコード相当の処理を実行する。

```
public List<Dept> findDeptByDeptName(String deptName) {
  return entityManager
    .createQuery("select d from Dept d where d.deptName = :deptName")
    .setParameter("deptName", deptName)
    .getResultList();
}
```

動的なクエリの自動生成

- エンティティのプロパティを条件とする検索（QueryByExample）

Daoを利用するコード

```
Dept example = new Dept();
example.setDeptNo(10);
List<Dept> list = dao.findByExample(example);
```

検索条件をセットしてDaoのメソッドに渡すだけ

```
SELECT dept FROM Dept AS dept WHERE (dept.deptNo = :deptNo)
```

自動生成されるクエリ

Daoを利用するコード

```
Dept example = new Dept();
example.setDeptNo(10);
example.setDeptName("hoge");
List<Dept> list = dao.findByExample(example);
```

```
SELECT dept FROM Dept AS dept WHERE ((dept.deptNo = :deptNo) AND
(dept.deptName = :deptName))
```

SQLのDTOへのマッピング

- SQLによる検索（QueryBySql）

hoge.foo.DeptDao

```
public List<DeptDto> findByDeptName(String deptName);
```

SQLファイルをDaoと同じパッケージに「*Dao名_メソッド名.sql*」の形式で用意

SQLコメントを利用して
メソッドの引数をバインドできる

hoge.foo.DeptDao_findByDeptName.sql

```
select id, dept_name from dept where dept_name = /*deptName*/'SALES'
```

テスト用の文字列

Daoを利用するコード

```
List<DeptDto> list = dao.findByDeptName("ACCOUNTING");
```

Q&A

おわり

ありがとうございました。