S2Dao入門

大中浩行(a.k.a.せとあずさ)

自己紹介

- 大中浩行(a.k.a.せとあずさ)
- azusa@fieldnotes.jp
- http://www.fieldnotes.jp/d/
- S2Dao/S2Container/S2Dao-CodeGen/mistralコミッタ
- （株）エルテックス SI事業部

agenda

- DBアプリケーションの問題点
- O/Rマッピング
- S2Daoとは
- S2Daoの特徴
- 関連ツール

DBアプリケーションの問題点

- 単調なコードの割りに実装量が多い
- 技術者がアプリケーション(Java)とDB(SQL)の両方に精通する必要があり、品質を保つのが難しい
- 大抵のアプリケーションではEJBは役不足
- とはいえ素のJDBCでは力不足

O/Rマッピング(Java)

- オブジェクトとデータベースのテーブルの結び付けを簡単にしようとする風潮
- Hibernate / iBATIS / Torque /Commons DBUtils(これはO/Rマッピングではないですが) / JDO(Java Data Object)/ TopLink / JPA(Java Presistance API)
- 世界的にはHibernateがデファクト？

S2Dao

- S2Dao
  - えすつーだお
  - http://s2dao.seasar.org/ja/
  - Daoパターンを使用したO/Rマッパー
  - AOPとアノテーションによってXMLレスなO/Rマッピングを実現
  - .NET(S2Dao.NET), PHP(S2Dao.PHP5)にも移植されています
  - 豊富なサポートツール

特徴

- Daoはインターフェースを書くだけで実装可能
- 設定はアノテーションで記述
- エンティティはJavaBean
- 単純なSQLは自動生成
- 複雑なSQLは自分で書く
- 2-Way SQL
  - S2Daoで実行するSQLがSQL*Plusなどで実行可能

Dao(Before)

```
public class EmployeeDao {
    public List<Employee> getAllEmployees() {
        Connection con = null;
        Statement stmt = null;
        try {
            List<Employee> list = new ArrayList<Employee>();
            Class.forName("org.hsqldb.jdbcDriver");
            con = DriverManager.getConnection(
                    "jdbc:hsqldb:hsql://localhost:9001", "sa", "");
            stmt = con.createStatement();
            ResultSet rset = stmt.executeQuery("select * from Employee");
            while (rset.next()) {
                Employee emp = new Employee();
                emp.setEmpno(rset.getInt("EMPNO"));
                emp.setEname(rset.getString("ENAME"));
                emp.setJob(rset.getString("JOB"));
                emp.setMgr(rset.getInt("MGR"));
                emp.setHiredate(rset.getDate("HIREDATE"));
                emp.setSal(rset.getBigDecimal("SAL"));
                emp.setComm(rset.getBigDecimal("COMM"));
                emp.setDeptno(rset.getInt("DEDPTNO"));
                emp.setTimestamp(rset.getTimestamp("TIMESTAMP"));
                list.add(emp);
            }
            return list;
        } catch (RuntimeException e) {
            throw e;
        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally {
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (Exception ignore) {
                }
            }
            if (con != null) {
                try {
                    con.close();
                } catch (Exception ignore) {
                }
            }
        }
    }
}
```

すみません、ちょっと小さすぎました

```
public class EmployeeDao {
    public List<Employee> getAllEmployees() {
        Connection con = null;
        Statement stmt = null;
        try {
            List<Employee> list = new ArrayList<Employee>();
            Class.forName("org.hsqldb.jdbcDriver");
            con = DriverManager.getConnection(
                    "jdbc:hsqldb:hsql://localhost:9001", "sa", "");
            stmt = con.createStatement();
            ResultSet rset = stmt.executeQuery("select * from
    Employee");
            while (rset.next()) {
                Employee emp = new Employee();
                emp.setEmpno(rset.getInt("EMPNO"));
                emp.setEname(rset.getString("ENAME"));
                emp.setJob(rset.getString("JOB"));
                emp.setMgr(rset.getInt("MGR"));
                emp.setHiredate(rset.getDate("HIREDATE"));
                emp.setSal(rset.getBigDecimal("SAL"));
                emp.setComm(rset.getBigDecimal("COMM"));
                emp.setDeptno(rset.getInt("DEDPTNO"));
                emp.setTimestamp(rset.getTimestamp("TIMESTAMP"));
                list.add(emp);
            }
            return list;
        } catch (RuntimeException e) {
            throw e;
        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally {
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (Exception ignore) {
                }
            }
            if (con != null) {
                try {
                    con.close();
                } catch (Exception ignore) {
                }
            }
        }
    }
}
```

Dao（After）

```
@S2Dao(bean=Employee.class)
public interface EmployeeDao{

   public List<Employee> getAllEmployees();

}
```

実装

- それではS2Daoを使用するコードを見てみましょう

- Employeeテーブル

カラム名	型	PK
EMPNO	numeric(4)	○
ENAME	varchar(10)	
JOB	varchar(9)	
MGR	numeric(7,2)	
HIREDATE	Date	
SAL	numeric(7,2)	
COMM	numeric(7,2)	
DEPTNO	numeric(2)	
TIMESTAMP	timestamp	

Dao

```
@S2Dao(bean=Employee.class)
public interface EmployeeDao{
    public List<Employee> getAllEmployees();
    @Arguments("empno")
    public Employee getEmployee(int
    empno);
    public void insert(Employee emp);
    public void delete(Employee emp);
    public void update(Employee emp);
}
```

エンティティ(JavaBean)

```
@Bean(table = "EMPLOYEE")
public class Employee implements Serializable {
(略)
    public Integer getEmpno() {
        return empno;
    }
    public void setEmpno(Integer empno) {
        this.empno = empno;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public String getJob() {
        return job;
    }
    public void setJob(String job) {
        this.job = job;
    }
    public Integer getMgr() {
        return mgr;
    }
    public void setMgr(Integer mgr) {
        this.mgr = mgr;
    }
    public Date getHiredate() {
        return hiredate;
    }
    public void setHiredate(Date hiredate) {
        this.hiredate = hiredate;
    }
    public BigDecimal getSal() {
        return sal;
    }
    public void setSal(BigDecimal sal) {
        this.sal = sal;
    }
    public BigDecimal getComm() {
        return comm;
    }
    public void setComm(BigDecimal comm) {
        this.comm = comm;
    }
    public Integer getDeptno() {
        return deptno;
    }
    public void setDeptno(Integer deptno) {
        this.deptno = deptno;
    }
    public Timestamp getTimestamp() {
        return timestamp;
    }
    public void setTimestamp(Timestamp tstamp) {
        this.timestamp = tstamp;
    }
    public String toString() {
        return org.apache.commons.lang.builder.ToStringBuilder
                .reflectionToString(this);
    }
}
```

定数アノテーション

- JDK1.4だとこうなります

```
public interface EmployeeDao{
    public static final Class BEAN = Employee.class;
    public List getAllEmployees();
    public static final String
                        getEmployee_ARGS="empno";
    public Employee  getEmployee(int empno);
    public void insert(Employee emp);
    public void delete(Employee emp);
    public void update(Employee emp);
}
```

Dao

- 命名規則
  - 挿入するメソッドは名前がinsert,create,addで始める
  - 更新するメソッドはupdate,modify,storeで始める
  - 削除するメソッドはdelete,~~update~~removeで始める
  - それ以外は検索

dicon

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD
  S2Container 2.4//EN"
"http://www.seasar.org/dtd/components24.dtd">
<components>
<include path="dao.dicon"/>
<component
  class="jp.fieldnotes.conf.dao.EmployeeDao" >
  <aspect>dao.interceptor</aspect>
</component>
</components>
```

実行コード

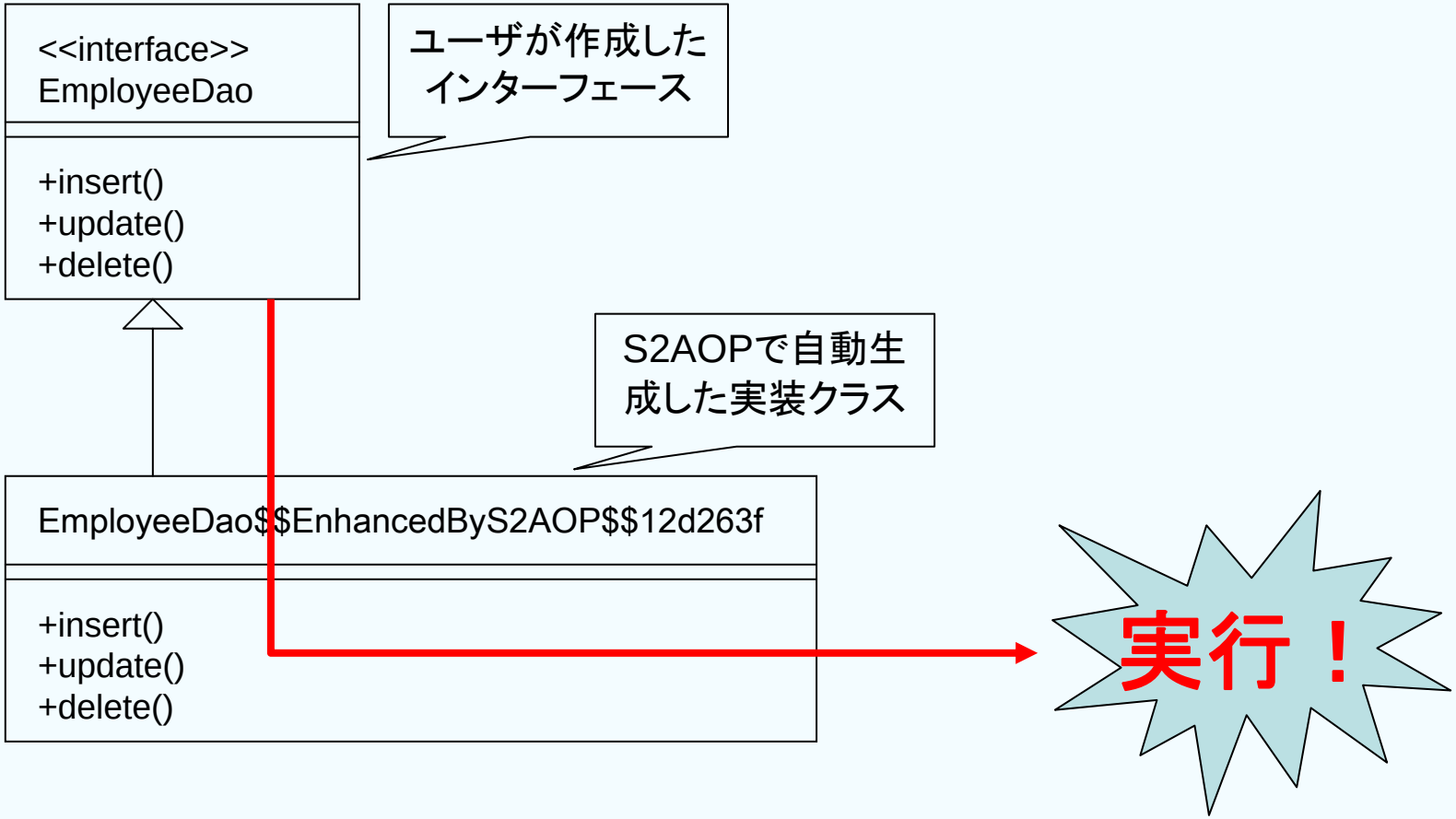
```
public static void main(String[] args) {
S2Container container = S2ContainerFactory.create("app.dicon");
try {
    container.init();
    EmployeeDao dao = (EmployeeDao) container
        .getComponent(EmployeeDao.class);
    System.out.println("daoを実行");
    List<Employee> result = dao.getAllEmployees();
    for (Employee employee : result) {
        System.out.println(employee);
    }
} finally {
    container.destroy();
}
}
```

実行！

```
daoを実行
DEBUG 2007-05-16 23:34:42,625 [main] 物理的なコネクションを取得しました
DEBUG 2007-05-16 23:34:42,625 [main] 論理的なコネクションを取得しました
DEBUG 2007-05-16 23:34:42,718 [main] 論理的なコネクションを閉じました
DEBUG 2007-05-16 23:34:42,718 [main] 論理的なコネクションを取得しました
DEBUG 2007-05-16 23:34:42,734 [main] 物理的なコネクションを取得しました
DEBUG 2007-05-16 23:34:42,734 [main] 論理的なコネクションを取得しました
DEBUG 2007-05-16 23:34:42,734 [main] 論理的なコネクションを閉じました
DEBUG 2007-05-16 23:34:42,906 [main] 論理的なコネクションを閉じました
DEBUG 2007-05-16 23:34:42,968 [main] SELECT EMPLOYEE.empno, EMPLOYEE.ename, EMPLOYEE.job, EMPLOYEE.mgr,
    EMPLOYEE.hiredate, EMPLOYEE.sal, EMPLOYEE.comm, EMPLOYEE.deptno, EMPLOYEE.timestamp FROM EMPLOYEE
DEBUG 2007-05-16 23:34:42,968 [main] 論理的なコネクションを取得しました
DEBUG 2007-05-16 23:34:43,000 [main] 論理的なコネクションを閉じました
jp.fieldnotes.conf.entity.Employee@f47396[empno=7369,ename=SMITH,job=CLERK,mgr=7902,hiredate=1980-12-17
    00:00:00.0,sal=800.00,comm=<null>,deptno=20,timestamp=2000-01-01 00:00:00.0]
jp.fieldnotes.conf.entity.Employee@b8f8eb[empno=7499,ename=ALLEN,job=SALESMAN,mgr=7698,hiredate=1981-02-20
    00:00:00.0,sal=1600.00,comm=300.00,deptno=30,timestamp=2000-01-01 00:00:00.0]
jp.fieldnotes.conf.entity.Employee@1de17f4[empno=7521,ename=WARD,job=SALESMAN,mgr=7698,hiredate=1981-02-22
    00:00:00.0,sal=1250.00,comm=500.00,deptno=30,timestamp=2000-01-01 00:00:00.0]
jp.fieldnotes.conf.entity.Employee@1f6ba0f[empno=7566,ename=JONES,job=MANAGER,mgr=7839,hiredate=1981-04-02
    00:00:00.0,sal=2975.00,comm=<null>,deptno=20,timestamp=2000-01-01 00:00:00.0]
jp.fieldnotes.conf.entity.Employee@1313906[empno=7654,ename=MARTIN,job=SALESMAN,mgr=7698,hiredate=1981-09-28
    00:00:00.0,sal=1250.00,comm=1400.00,deptno=30,timestamp=2000-01-01 00:00:00.0]
DEBUG 2007-05-16 23:34:43,031 [main] 物理的なコネクションを閉じました
DEBUG 2007-05-16 23:34:43,031 [main] 物理的なコネクションを閉じました
```

秘密兵器:AOP

- インターフェースしか用意してないのに処理が行われるのはどうして？



2Way-SQL(1)

```
@S2Dao(bean=Employee.class)
public interface EmployeeDao{
    public List getAllEmployees();
@Arguments("empno")
public Employee getEmployee(int empno);
    public void insert(Employee emp);
    public void delete(Employee emp);
    public void update(Employee emp);

}
```

2Way-SQL(2)

- 対話型ツールからそのまま実行可能
  - SELECT * FROM EMPLOYEE
    WHERE empno = /*empno*/7369
- ファイル名は「Daoのクラス名_メソッド名.sql」
  - この場合EmployeeDao_getEmployee.sql
  - Eclipseの場合はソースファイルと同じ場所に置く
  - Maven2の場合はsrc/main/resources

2Way-SQL(3)

- S2Daoから実行するときはSQLコメントの記述を元にPreparedStatementを生成して実行

```
SELECT * FROM Employee
    WHERE empno = /*empno*/7788
```

↑/*の後ろにスペースは入れない

→SELECT * FROM Employee WHERE empno = ?

というPreparedStatementが生成されます。

動的SQL

- 職種および部署を指定して検索する

```
@Arguments( { "job", "deptno" })
public List<Employee>
  getEmployeeByJobAndDeptno(String job,
  Integer deptno);
```

動的SQLとSQLコメント

```
SELECT * FROM EMPLOYEE
/*BEGIN*/WHERE
  /*IF job != null*/
     JOB = /*job*/'CLERK'
  /*END*/
  /*IF deptno != null*/
     AND DEPTNO =/*deptno*/20
  /*END*/
/*END*/
```

実行コード

```
System.out.println("daoを実行");
// 第1引数→job 第2引数→deptno
List<Employee> result =
  dao.getEmployeeByJobAndDeptno(null, 20);
for (Employee employee : result) {
  System.out.println(employee);
}
```

実行！

```
daoを実行
DEBUG 2007-05-16 23:35:41,765 [main] 物理的なコネクションを取得しました
DEBUG 2007-05-16 23:35:41,765 [main] 論理的なコネクションを取得しました
DEBUG 2007-05-16 23:35:41,859 [main] 論理的なコネクションを閉じました
DEBUG 2007-05-16 23:35:41,859 [main] 論理的なコネクションを取得しました
DEBUG 2007-05-16 23:35:41,875 [main] 物理的なコネクションを取得しました
DEBUG 2007-05-16 23:35:41,875 [main] 論理的なコネクションを取得しました
DEBUG 2007-05-16 23:35:41,875 [main] 論理的なコネクションを閉じました
DEBUG 2007-05-16 23:35:42,031 [main] 論理的なコネクションを閉じました
DEBUG 2007-05-16 23:35:42,093 [main] SELECT * FROM EMPLOYEE
WHERE

  DEPTNO =20

DEBUG 2007-05-16 23:35:42,093 [main] 論理的なコネクションを取得しました
DEBUG 2007-05-16 23:35:42,125 [main] 論理的なコネクションを閉じました
jp.fieldnotes.conf.entity.Employee@b8f8eb[empno=7369,ename=SMITH,job=CLERK,mgr=7902,hiredate=19
    80-12-17 00:00:00.0,sal=800.00,comm=<null>,deptno=20,timestamp=2000-01-01 00:00:00.0]
jp.fieldnotes.conf.entity.Employee@1f6ba0f[empno=7566,ename=JONES,job=MANAGER,mgr=7839,hiredate
    =1981-04-02 00:00:00.0,sal=2975.00,comm=<null>,deptno=20,timestamp=2000-01-01 00:00:00.0]
DEBUG 2007-05-16 23:35:42,156 [main] 物理的なコネクションを閉じました
DEBUG 2007-05-16 23:35:42,156 [main] 物理的なコネクションを閉じました
```

テスト

- テストの重要性は言わずもがな
- でもDBが絡むテストはテストが難しい…
- テストデータの準備が…
- 環境を担当者ごとに用意するのは…
- 結果の検証が…
- そんなあなたにS2Unit(S2DaoTestCase)

S2Unit(S2DaoTestCase)

- Excelでテストデータ/期待値を準備してテストできる
- テスト終了後にテーブルをロールバックするため、DBを共有している環境でもテスト可能！

```
public void testGetEmployeeTx() throws Exception
{
    readXlsAllReplaceDb("EmployeeDaoTest.xls");
    Employee result = dao.getEmployee(7369);
    assertBeanEquals("7369",
    readXls("EmployeeDaoTest_result.xls"), result);
}
```

こんなときは

- 単純なリレーションくらい自動生成してほしいよ
  - N:1マッピングは自動生成可能です。
- トランザクション制御は？

  Seasar2のトランザクションの自動制御機能を使って、**Daoを呼び出すコンポーネントに**トランザクションをかけます。
- IDを自動生成しているんですけど
  - BeanのPKに対応するプロパティにアノテーションをつけます。(IDアノテーション)
- 排他処理
  - タイムスタンプおよびバージョン番号を使用する楽観的排他をサポートしています。

豊富なサポートツール

- S2Daoプラグイン
  - Daoのメソッドに対応したSQLファイルを開く・作成する
- DBFlute
  - Dao/Emtityをスキーマから自動生成
    →詳しくは17:00からのセッションで
- S2Dao-CodeGen
  - テーブル定義書からDao/Dtoを自動生成
  - スキーマからの生成もできます。そのうち
    →**コミッタ募集中！**
- Dolteng
  - Eclipseプロジェクトを作成するEclipseプラグイン
  - Entity/Daoをスキーマから自動生成

まとめ：S2Daoとは

- XMLレス、実装不要なO/Rマッパー
- Daoはインターフェースのみ
- 単純なSQLは自動生成、複雑なSQLは~~自動~~手動生成
- もっと詳しく
  - https://www.seasar.org/svn/s2container/trunk/seasar2-tutorial/doc/S2Dao.ppt

ご意見・ご質問は

- 質問はSeasar-Userメーリングリストまで
  - https://ml.seasar.org/mailman/listinfo/seasar-user
- 開発に関する議論はseasar-s2dao-devメーリングリストまで
  - https://ml.seasar.org/mailman/listinfo/seasar-s2dao-dev

ありがとうございました。