

D B フレームワーク for Java  
dbfw ver1.0.1

## 1. ファイルの構成

dbfw101.jar (DBフレームワーク本体)

db.properties (DB接続情報、SQL文のプロパティファイル)

## 2. db.properties ファイルの記述

db.properties ファイルはクラスパスをしている場所においてください。

### 2.1 Log クラスの指定

```
*****  
# LOG_CLASS  
*****  
LOG_CLASS=sample01.Log
```

アプリケーションで使う Log クラスを指定してください。Log クラスは、net.core22.dbfw.Log を implements して作成してください

### 2.2 DB 接続情報の設定

DB の接続には、DataSource を使わない方法と DataSource を使う方法に分かれます。前者はクライアントアプリケーション、サーブレットで、後者はサーブレットで主に使われます。Web アプリケーションサーバー (Tomcat 等) を用いる場合は、DataSource を使う方法を用いる方法がよく使われます。

#### 2.2.1 DataSource を使わない方法

ORACLE の例

```
*****  
# DB Driver  
*****  
Mode=0  
User=userid  
Password=password  
Driver=oracle.jdbc.driver.OracleDriver  
URL=jdbc:oracle:thin:@127.0.0.1:1521:oradb
```

DataSource を使わない場合、Mode は 0 を指定してください。

## SQLite の場合

```
#####  
# DB Driver  
#####  
Mode=0  
User=  
Password=  
Driver=org.sqlite.JDBC  
URL=jdbc:sqlite:/C:/user/workspace/sample01/lib/sample01.db
```

### 2.2.2 DataSource を使う方法

#### ORACLE の場合

```
#####  
# DB DataSource  
#####  
Mode=1  
LookUpURL = java:comp/env/jdbc/ORACLE
```

DataSource を使う場合、Mode は 1 を指定してください。

#### MySQL の場合

```
#####  
# DB DataSource  
#####  
Mode=1  
LookUpURL=java:comp/env/jdbc/MYSQL
```

### 2.3 SQL 文の指定

```
SQL001=select id,name,sex from user where (id=? or id=?) and sex=?  
SQL002=select id,name from categories where id=?  
SQL003=insert into user(id,name,sex) values(?,?,?)  
SQL004=delete from user where id=?
```

KEY は SQL で始まる必要はありません。プログラムから呼び出すときの名前を指定してください。? はプログラムから値を渡すことを示します。また、ここに書かずにプログラムに直接 SQL 文を書くことも可能です。

### 3. プログラムの作成

#### 3.1 DataSource を使わない方法

```
public static void main(String[] args) throws ValidateException,
SQLException, DBException {
    DB db = new DB();
    try {    db.connect();
            //SQL に渡すパラメータ
            Param param = new Param();
            param.add(0, "0001");
            param.add(1, "0004");
            param.add(2, "0");
//SELET 文を発行する
            //UserEntity クラスで、EntityList として取得できる

            EntityListelist=db.executeQuery("SQL001",param,UserEntity.class);
            //データ読み出し
            int size = elist.size();
            for (int i = 0; i < size; i++) {
                UserEntity en = (UserEntity)elist.get(i);
                System.out.println("id="+en.getId()+"
name="+en.getName());
            }
        }catch(SQLException e){
            throw e;
        }catch(DBException e){
            throw e;
        }catch(ValidateException e){
            throw e;
        }finally{
            try{
                db.close();
            }catch(Exception ex){
                ex.printStackTrace();
            }
        }
    }
}
```

### 3.2 DataSource を使う方法

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    DataSource ds = null;
    DB db = null;
    try {
        ds = DataSourceFactory.getDataSource();
        db = new DB(ds);
        db.connect();
        //SQL に渡すパラメータ
        Param param = new Param();
        param.add(0, "1");
        //SELET 文を発行する
        //UserEntity クラスで、EntityList として取得できる
        EntityList elist =
db.executeQuery("SQL002",param,DataEntity.class);
        //データ読み出し
        int size = elist.size();
        for (int i = 0; i < size; i++) {
            DataEntity en = (DataEntity)elist.get(i);
            out.println("id="+en.getId()+"
name="+en.getName());
        }
    }catch(SQLException e){
        e.printStackTrace();
    }catch(DBException e){
        e.printStackTrace();
    }catch(ValidateException e){
        e.printStackTrace();
    }finally{
        try{
            db.close();
        }catch(Exception ex){
            ex.printStackTrace(); }}}}
```

### 3.2 Log クラスの実装

```
public class Log implements net.core22.dbfw.Log {  
    /**  
     * ログ ( debug ) 出力  
     *  
     * @param msg メッセージ  
     */  
    public void debug(String msg) {  
        System.out.println("LOG>" + msg);  
    }  
    /**  
     * ログ ( error ) 出力  
     *  
     * @param msg メッセージ  
     */  
    public void error(String msg) {  
        System.out.println("LOG>" + msg);  
    }  
    /**  
     * ログ ( fatal ) 出力  
     *  
     * @param msg メッセージ  
     */  
    public void fatal(String msg) {  
        System.out.println("LOG>" + msg);  
    }  
    /**  
     * ログ ( info ) 出力  
     *  
     * @param msg メッセージ  
     */  
    public void info(String msg) {  
        System.out.println("LOG>" + msg);  
    }  
}
```

### 3.3 Entity クラスの実装

#### テーブル (ビュー)

フィールドの編集

	フィールド名	データ型	NULL	重複	インデックス	フィールドコメント
↑	id	文字列型	OK	NG	<input type="checkbox"/>	
	name	文字列型	OK	OK	<input type="checkbox"/>	
↓	sex	整数型	OK	OK	<input type="checkbox"/>	0:男性 1:女性

エディター: id      初期値:      フィールドの追加      キャンセル

```
public class UserEntity extends Entity implements Validate{
    public String getId() throws DBException{
        return (String)this.getObject("id");
    }
    public void setId(String id){
        this.setObject("id",id);
    }
    public String getName() throws DBException{
        return (String)this.getObject("name");
    }
    public void setName(String name){
        this.setObject("id",name);
    }
    public String getSex() throws DBException{
        return (String)this.getObject("sex");
    }
    public void setSex(String sex){
        this.setObject("sex",sex);
    }
    public ValidateErrors validate() throws ValidateException {
        // SELECT 時の値の妥当性チェックを書く
        // DB#executeQuery で、これを実装しているところのメソッドがそのとき実行される。
        // ValidateErrors にエラーの内容を入れる
        ValidateErrors ers = new ValidateErrors();
        return ers;
    }
}
```

### 3.4 INSERT、DELETE、UPDATE

#### INSERT 例

```
db.connect();
Param param01 = new Param();
param01.add(0, "0004");
param01.add(1, "山田 次郎");
param01.add(2, "0");
db.executeUpdate("SQL003",param01);
db.commit();
db.close();
```

明示的に commit,rollback をしてください。また必ず、db.close()の処理を行うようにしてください。

#### DELETE 例

```
Param param02 = new Param();
param02.add(0, "0004");
db.executeUpdate("SQL004",param02);
db.commit();
db.close();
```

明示的に commit,rollback をしてください。また必ず、db.close()の処理を行うようにしてください。

### 3.5 SQL 文をプログラムから文字列で指定する方法

DB#executeQuery(java.lang.String sqlString, java.lang.Class cls)

DB#executeUpdate(java.lang.String sqlString)

を使うと、SQL 文を db.properties に定義しないで SQL を実行することが可能です。



#### 4. 補足事項

net.core22.dbfw.DBException: db.properties が見つからないか、指定した KEY が存在しません (key=LOG\_CLASS)

##### 対応方法

db.properties の位置を確認してください。クラスパスの通っているところにおいてください。

名前 jdbc はこのコンテキストにバインドされていません  
javax.naming.NameNotFoundException: 名前 jdbc はこのコンテキストにバインドされていません

##### 対応方法

JDBC をクラスパスの通っているところにおいてください。

または、JNDI のリソースの設定 (データソースを使用する場合) が正しくない可能性があります。