Mapper & SQL

- Official Document : http://www.mybatis.org/mybatis-3/getting-started.html
- Reference : https://jirafnd.dev.activenetwork.com/browse/ANE-37818
  - create mapper interface

```java
package com.activenet.mybatis.mappers;

import java.util.List;

import ActiveNetLib.Tools.MapParam;
import ActiveNetLib.Tools.MapRecord;

public interface PackageMapper {

    List<MapRecord> findPackages(MapParam param);

}
```

  - create xml for mapper

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.activenet.mybatis.mappers.PackageMapper">
    <select id="findPackages" resultType="ActiveNetLib.Tools.MapRecord" statementType="CALLABLE">
        {call search_packages(
            #{package_name, jdbcType=VARCHAR,mode=IN },
            #{category_id,jdbcType=INTEGER,mode=IN},
            #{site_id,jdbcType=INTEGER,mode=IN},
            #{status_id,jdbcType=INTEGER,mode=IN},
            #{entry_point_id,jdbcType=INTEGER,mode=IN},
            #{available_as_prerequisite,jdbcType=INTEGER,mode=IN},
            #{retention_eligible,jdbcType=INTEGER,mode=IN}
        )}
    </select>
</mapper>
```

  - append xml to mybatis-config.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <mappers>
        <mapper resource="com/activenet/mybatis/mappers/PackageMapper.xml"/>
    </mappers>
```

```
</configuration>
```

SqlSession & Transaction

- SqlSession open/close by ANSqlSessionManager
- ANSqlSessionManager instance hold by OrgContext.getSQLSessionManager()

```
SqlSession session = OrgContext.getSQLSessionManager().openSession(dbc);
OrgContext.getSQLSessionManager().closeSession(session);
```

  - Note1: SqlSession must be opened by DBConnection to avoid cross transaction, refer to https://jirafnd.dev.activenetwork.com/browse/ANE-39686
  - Note2: SqlSession just used for sql statement executor, it does not manage transaction(commit & rollback), refer to https://jirafnd.dev.activenetwork.com/browse/ANE-39686
  - Note3: DBConnection followed the JDBC specification, refer to https://jirafnd.dev.activenetwork.com/browse/ANE-41434
- Transaction managed by DBConnectionManager

```
  DBConnection dbc = null;
  SqlSession session = null;
  try {
    dbc = dbcm.getConnection();
    session = OrgContext.getSQLSessionManager().openSession(dbc);
    // call store procedure by mybatis
  } catch (Exception e) {
    // some thing error and need to rollback
  } finally {
    OrgContext.getSQLSessionManager().closeSession(session);
    dbcm.freeConnection(dbc);
  }
```

  - Note: ANTools.SqlErrorInterceptor will set DBConnection.sql_error = true if callable statement execute failed.
  - Refer: https://jirafnd.dev.activenetwork.com/browse/ANE-41892
  - Another complex usage:

```
  DBConnection dbc = null;
  // get connection from dbcm and begin a transaction
  try {
    dbc = dbcm.getConnection();
    // access database by dbc
  } catch (Exception e) {
    // resolve exception
  }
  SqlSession session = null;
  try {
    session = OrgContext.getSQLSessionManager().openSession(dbc);
    // call store procedure by mybatis
```

```java
    } catch (Exception e) {
      // some thing error and need to rollback
    } finally {
      OrgContext.getSQLSessionManager().closeSession(session);
    }
    try {
      SomeClass.someStaticMethod(dbc);
      someInstance.someMethod(dbc);
    } catch (Exception e) {
      // resolve exception
    }
    // first transaction will be commit if no error
    dbcm.freeConnection(dbc);

    // need to re-get connection from dbcm, and another transaction begin
    try {
      dbc = dbcm.getConnection();
      // access database by dbc
    } catch (Exception e) {
      // resolve exception
    }
```

Parameter

- create new class, eg, PackageQuery for searching package.
- use ActiveNetLib.Tools.MapParam

```java
MapParam param = new MapParam();
param.put("param_name", 8); // int
param.put("param_name", 8); // long
param.put("param_name", 0.52); // double/float
param.put("param_name", "city"); // string
param.put("param_name", true); // bool
param.put("param_name", new BigDecimal(2016.0108)); // decimal
param.put("param_name", new int[] { 0, 1, 2 }); // int array
param.put("param_name", Arrays.asList(0, 1, 2)); // int collection
param.put("param_name", new byte[] {}); // byte array
param.put("param_name", new Date()); // datetime
param.put("param_name", new Date()); // date
param.put("param_name", new Date()); // time
param.put("param_name", new Object()); // others

SomeMapper mapper = session.getMapper(SomeMapper.class);
SomeType result = mapper.someMethod(param);
```

  - Note1: key is case insensitive in MapParam

- Note2: type of bool in sql server is bit, and true = 1 & false = 0, but in ANet DBConnection.DB_TRUE = -1 & DBConnection.DB_FALSE = 0
  - ActiveNetLib.Tools.MapParam.putBoolean(String, Boolean)
- Note3: refer to https://jirafnd.dev.activenetwork.com/browse/ANE-41041
- Note4: refer to https://jirafnd.dev.activenetwork.com/browse/ANE-41410
- Note5: string encode/decode & escaped the keyword/wildcard of sql server
  - Decode & !Escape : MapParam.putString(String, String)
    - eg. keyword searching
  - Decode & Escape : MapParam.putStringEscaped(String, String)
    - Note: input string include [* ? _ % will be escaped
  - MapParam.put(String, Object)
    - original input string will pass in mybatis
- Note6: where in clause
  - sql like ' where id in (@ids) ', there are two ways below:
    - MapParam.putInList(String, int[])
    - MapParam.putInList(String, Collection<Integer>)
    - Note: In store procedure, @ids should be varchar
- Note7: enum, pass in enum's ordinal actually
  - MapParam.putEnum(String, Enum<?>)

Result Mapping

- official reference : http://www.mybatis.org/mybatis-3/sqlmap-xml.html#Result_Maps
- Note: refer to https://jirafnd.dev.activenetwork.com/browse/ANE-41041
- Note: refer to https://jirafnd.dev.activenetwork.com/browse/ANE-41410
- compatible usage: ActiveNetLib.Tools.MapRecord

```
<select resultType="ActiveNetLib.Tools.MapRecord">
    <!-- sql statemtn -->
</select>
```

```
List<MapRecord> records = mapper.findPackages(param);
if (records != null && !records.isEmpty()) {
    for (MapRecord record : records) {
        packages.add(new Package(record));
    }
}
```

- Note: both of MapRecord and DBRecord implemented ActiveNetLib.Tools.Record, so change constructor from new Package(DBRecord dbr) to new Package(Record dbr)

```
public class Package {
 // public Package(DBRecord dbr) {
```

```java
  public Package(Record dbr) {
    this.field_name = dbr.getString("field_name");
    // others
  }
}
```

Q&A

- we need to use jdbc type correctly, details below:

```java
package org.apache.ibatis.type;

import java.sql.Types;
import java.util.HashMap;
import java.util.Map;

/**
 * @author Clinton Begin
 */
public enum JdbcType {
  /*
   * This is added to enable basic support for the
   * ARRAY data type - but a custom type handler is still required
   */
  ARRAY(Types.ARRAY),
  BIT(Types.BIT),
  TINYINT(Types.TINYINT),
  SMALLINT(Types.SMALLINT),
  INTEGER(Types.INTEGER),
  BIGINT(Types.BIGINT),
  FLOAT(Types.FLOAT),
  REAL(Types.REAL),
  DOUBLE(Types.DOUBLE),
  NUMERIC(Types.NUMERIC),
  DECIMAL(Types.DECIMAL),
  CHAR(Types.CHAR),
  VARCHAR(Types.VARCHAR),
  LONGVARCHAR(Types.LONGVARCHAR),
  DATE(Types.DATE),
  TIME(Types.TIME),
  TIMESTAMP(Types.TIMESTAMP),
  BINARY(Types.BINARY),
  VARBINARY(Types.VARBINARY),
  LONGVARBINARY(Types.LONGVARBINARY),
  NULL(Types.NULL),
  OTHER(Types.OTHER),
  BLOB(Types.BLOB),
  CLOB(Types.CLOB),
  BOOLEAN(Types.BOOLEAN),
  CURSOR(-10), // Oracle
  UNDEFINED(Integer.MIN_VALUE + 1000),
  NVARCHAR(Types.NVARCHAR), // JDK6
  NCHAR(Types.NCHAR), // JDK6
  NCLOB(Types.NCLOB), // JDK6
  STRUCT(Types.STRUCT);

  public final int TYPE_CODE;
  private static Map<Integer,JdbcType> codeLookup = new HashMap<Integer,JdbcType>();

  static {
    for (JdbcType type : JdbcType.values()) {
      codeLookup.put(type.TYPE_CODE, type);
    }
  }

  JdbcType(int code) {
    this.TYPE_CODE = code;
  }

  public static JdbcType forCode(int code)  {
    return codeLookup.get(code);
  }

}
```