# 数据库方言插件开发

## 前言

我们已经支持包括MySQL，Oracle，Postgre等许多常用数据库，但某些时候，客户需要使用某些我们不支持的数据库，也就是我们没有提供对应的数据库操作的方法，好在我们提供了用于处理数据库之间差异的接口（方言），这个时候就需要开发单独的插件实现这个接口的某些方法来处理特定需求。

第一步：首先我们需要建立数据库方言对象生成接口的实现类，该接口提供了两个方法返回对应的数据库方言类，详见下图注释。

### 数据库方言对象生成接口

DialectCreator

```
package com.fr.stable.fun;

import com.fr.stable.UrlDriver;
import com.fr.stable.fun.mark.Mutable;

import java.sql.Connection;

/**
 *
 */
public interface DialectCreator extends Mutable {

    String XML_TAG = "DialectCreator";

    int CURRENT_LEVEL = 1;

    /**
     *
     * driver, return null
     *
     * @param driver
     * @return
     */
    Class<?> generate(UrlDriver driver);

    /**
     * null
     *
     * @param connection
     * @return
     */
    Class<?> generate(Connection connection);

}
```

在按照规范实现编写了实现这个接口的类后，在插件配置文件中注册这个类即可

Demo

```
<extra-core>
    <DialectCreator class="com.fr.plugin.testDialect.DialectCreatorImpl"/>
</extra-core>
```

示例实现(在该示例中，我们假定在urlDriver为“com.mysql.jdbc.Driver”时返回对应的数据库方言实现类TestDialect类，也可根据Connection来判断何时使用插件提供的数据库方言实现类)

testDialect

```
package com.fr.plugin.adsDialect;
import com.fr.general.ComparatorUtils;
import com.fr.stable.UrlDriver;
import com.fr.stable.fun.impl.AbstractDialectCreator;
import java.sql.Connection;

public class DialectCreatorImpl extends AbstractDialectCreator{
    @Override
    public int currentAPILevel() {
        return CURRENT_LEVEL;
    }
    @Override
    public Class<?> generate(UrlDriver driver) {
        if (ComparatorUtils.equals(driver.getDriver(),
"com.mysql.jdbc.Driver")){
            return TestDialect.class;
        }
        //return null, metadata, drivermetadata, .
        //driver, metadata.
        return null;
    }
    @Override
    public Class<?> generate(Connection connection) {
        return null;
    }
}
```

第二步：实现对应的数据库方言实现类，也就是TestDialect。

数据库方言接口，也就是用于处理数据库之间差异的接口（该类型插件开发最重要的接口）

Dialect

```
package com.fr.data.core.db.dialect;

import com.fr.base.StoreProcedureParameter;
import com.fr.data.core.db.ColumnInformation;
```

```
import com.fr.data.core.db.TableProcedure;
import com.fr.data.core.db.dml.Table;
import com.fr.data.core.db.handler.SQLTypeHandlerFactory;
import com.fr.data.core.db.tableObject.Column;
import com.fr.data.dao.JDBCDataAccessObjectOperator;
import com.fr.data.dao.ObjectMappingTable;

import javax.transaction.NotSupportedException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;

/**
 *
 */
public interface Dialect {

    public static final int DISABLE_FOREIGN_KEY_CHECKS = 0;
    public static final int ENABLE_FOREIGN_KEY_CHECKS = 1;

    /**
     *
     *
     * @param manager
     * @param cn
     * @param tableObj
     * @throws Exception
     */
    public void initTables(JDBCDataAccessObjectOperator manager, Connection
cn, ObjectMappingTable[] tableObj) throws Exception;


    /**
     * NULL
     *
     * @param columnSQL sql
     * @param column
     * @return columnSQL
     */
    public StringBuffer columnInit(StringBuffer columnSQL, Column column);

    /**
     *
     *
     * @param delCascade
     * @param dialect
     * @param buf
     * @return
     */
    public String cascadeDeletePosition(boolean delCascade, boolean
isSupportsCascadeDelete, StringBuffer buf);
```

```
    /**
     *
     *
     * @return
     */
    public int getFetchSize();

    /**
     *
     *
     * @param columnName
     * @return
     */
    public String column2SQL(String columnName);

    /**
     * where
     *
     * @param string
     * @param type
     * @return
     */
    public String column2SQL4WhereSQL(String string, int type);

    /**
     *
     *
     * @param columnType
     * @param columnSize
     * @return
     */
    public String columnType2SQL(int columnType, String columnSize);

    /**
     * SQL
     *
     * @param table
     * @return SQL
     */
    public String table2SQL(Table table);

    /**
     *
     *
     * @param conn
     * @return
     * @throws Exception
     */
    public String[] getSchemas(Connection conn) throws Exception;

    /**
```

```
     *
     *
     * @param connection
     * @param schema
     * @param b          oracle
     * @return
     * @throws Exception
     * @deprecated charset
     */
    public TableProcedure[] getTableProcedure(Connection connection, String
schema, boolean b) throws Exception;

    /**
     *
     *
     * @param database
     * @param connection
     * @param schema
     * @param b          oracle
     * @return
     * @throws Exception
     */
    public TableProcedure[] getTableProcedure(com.fr.data.impl.Connection
database, Connection connection, String schema, boolean b) throws
Exception;

    /**
     *
     *
     * @param connection
     * @param result
     * @param catalog
     * @param schema
     * @return
     */
    public TableProcedure[] getProcedureList(Connection connection,
ResultSet result, String catalog, String schema);

    /**
     * SQL
     *
     * @param conn
     * @param tableName
     * @param columnName
     * @param columnType
     * @return SQL
     */
    public String createSequence(Connection conn, String tableName, String
columnName, String columnType);

    /**
     * select
     *
```

```
     * @param table
     * @param column
     * @param type
     * @return select
     * @throws Exception
     */
    public String getIdentitySelectString(String table, String column, int
type) throws Exception;

    /**
     *
     *
     * @param conn
     * @param autoCommit truefalse
     * @throws SQLException
     */
    public void setAutoCommit(Connection conn, boolean autoCommit) throws
SQLException;

    /**
     *
     *
     * @return truefalse
     */
    public boolean supportsLimitOffset();

    /**
     * SQL
     *
     * @param query  SQL
     * @param offset
     * @param limit
     * @return SQL
     */
    public String getLimitString(String query, int offset, int limit);

    /**
     * SQL
     *
     * @param sql
     * @return
     */
    public String getCountSql(String sql);


    /**
     * MySQL
     *
     * @param conn
     * @param type
     * @param table
     * @param columnName
     * @return truefalse
```

```
     */
    public boolean isYearData(Connection conn, int type, Table table,
String columnName);

    /**
     *
     *
     * @return
     */
    public String quartzDelegateClass();

    /**
     * key
     *
     * @return key
     */
    public String quartzDelegateKey();

    /**
     *
     *
     * @param conn
     * @return SQL
     */
    public String defaultValidationQuery(Connection conn);

    /**
     *
     *
     * @param conn
     * @param name
     * @return
     */
    public StoreProcedureParameter[]
getStoreProcedureDeclarationParameters(Connection conn, String name, String
parameterDefaultValue);

    /**
     * SQL
     *
     * @return SQL
     */
    public SQLTypeHandlerFactory buildSQLTypeHandlerFactory();

    /**
     *
     *
     * @return truefalse
     */
    public boolean supportsCascadeDelete();

    /**
     *
```

```
     *
     * @param constraintName
     * @param foreignKey
     * @param referencedTable
     * @param primaryKey
     * @param referencesPrimaryKey
     * @return SQL
     */
    public String buildForeignKeyString(
            String constraintName,
            String[] foreignKey,
            String referencedTable,
            String[] primaryKey,
            boolean referencesPrimaryKey);

    /**
     *
     *
     * @return truefalse
     */
    public boolean supportsUniqueConstraintInCreateAlterTable();

    /**
     *
     *
     * @return truefalse
     */
    public boolean supportsUniqueViolationExceptionCheck();

    /**
     * SQL
     *
     * @param se SQL
     * @return truefalse
     */
    public boolean isUniqueViolationException(SQLException se);

    /**
     *
     *
     * @param conn
     * @param name
     * @param charSetName
     * @return
     */
    public String getStoreProcedureText(java.sql.Connection conn, String
name, String charSetName);

    /**
     *
     *
     * @return true
     */
```

```
    public boolean isSupportFetchText();

    /**
     * field
     *
     * @param value
     * @param type
     * @return
     */
    public Object parseValue(Object value, int type);

    /**
     *
     *
     * @param conn
     * @param tableName
     * @param schema
     * @return
     */
    public String getTableCommentName(java.sql.Connection conn, String
tableName, String schema, String dbLink);

    /**
     *
     *
     * @param conn
     * @param tableName
     * @param schema
     * @return
     */
    public List getTableFieldsInfor(java.sql.Connection conn, String
tableName, String schema, String dbLink);

    /**
     *
     *
     * @param conn
     * @return
     */
    public Statement createLimitUseStatement(java.sql.Connection conn)
throws SQLException;

    /**
     *
     * FIXME Sqlserver  endRow
     *
     * @param database
     * @param conn
     * @param stmt
     * @param fieldNames
     * @param schema
     * @param tableName
     * @param dbLink
```

```
     * @param startRow
     * @return
     */
    public ResultSet createLimitResultSet(com.fr.data.impl.Connection
database,
                                          java.sql.Connection conn,
Statement stmt,
                                          String[] fieldNames, String
schema,
                                          String tableName, String dbLink,
long startRow) throws SQLException;

    /**
     * SQL
     *
     * @param sql        sql
     * @param fieldNames
     * @param startRow
     * @return SQL
     */
    public String createLimitSQL(String sql, String[] fieldNames, long
startRow) throws NotSupportedException;

    /**
     * row
     *
     * @param sql sql
     * @param row
     * @return
     */
    public String getSpecificRowSql(String sql, int row);

    /**
     * startrow
     *
     * @param sql   sql
     * @param start
     * @param end
     * @return
     */
    public String getRowRangeSql(String sql, int start, int end, String[]
cols);

    /**
     * rowCount
     *
     * @param rowCount
     * @param table
     * @return
     * @throws NotSupportedException
     */
    public String getTopNRowSql(int rowCount, Table table) throws
NotSupportedException;
```

```
    /**
     *
     *
     * @param insertCount
     * @return
     */
    public boolean isSupportQueryWhileInsert(int insertCount);

    /**
     *
     *
     * @param connection
     * @param rs
     * @param oriCharsetName
     * @param newCharsetName
     * @return
     * @throws SQLException
     */
    public ColumnInformation[] getColumnInformation(Connection connection,
ResultSet rs, String sql, String oriCharsetName, String newCharsetName)
throws SQLException;

    /**
     *
     *
     * @param conn
     * @param keyChecks
     *                   DISABLE_FOREIGN_KEY_CHECKS ,
ENABLE_FOREIGN_KEY_CHECKS
     */
    public void setForeignKeyChecks(Connection conn, int keyChecks);

    /**
     * sql
     *
     * @param query sql
     * @return sql
     */
    String createSQL4Columns(String query);

    /**
     *
     *
     * @return
     */
    public TableProcedure[]
getAllTableProcedure(com.fr.data.impl.Connection database, String type);

    /**
     *
     *
     * @param connection
```

```
     * @param sql        sql
     * @return
     */
    public Statement createStatement(Connection connection, String sql)
throws SQLException;

    /**
     * sql
     *
     * @param sql
     * @param statement
     * @param connection
     * @return
     */
    public ResultSet executeQuery(Statement statement, String sql,
Connection connection) throws SQLException;

    /**
     *
     *
     * @param object
     * @return
     */
    public boolean isNULL(Object object);

    /**
     *
     *
     * @param cn
     * @param table
     * @throws Exception
     */
    public void notifyTriggerChange(Connection cn, Table table,
TriggerAction action);

    /**
     * RPC:
     *
     * @param connection
     * @param query      SQL
     * @return Object[2] ~ [Statement, ResultSet]
     * @date 2014-9-30-5:11:47
     */
    public Object[] remoteProcedureCall(Connection connection, String
```

```
query) throws SQLException;

}
```

各个方法都给了相关注释，用于处理数据库差异的方方面面，这里以阿里云ADS上的MySQL为例（用odbc直连的时候获取不到所有表，还不提供API获取原始数据库信息，只能反射去拿odbc里的驱动文件名来判断不同数据库，再写不同的sql获取表），实现该接口获取所有表的方法(一般来说，不需要直接实现com.fr.data.core.db.dialect.Dialect接口，只要继承com.fr.data.core.db.dialect.AbstractDialect抽象类即可。)。

AdsMysqlOdbcDialect

```
package com.fr.plugin.adsDialect;

import com.fr.base.FRContext;
import com.fr.data.core.db.ColumnInformation;
import com.fr.data.core.db.DBUtils;
import com.fr.data.core.db.TableProcedure;
import com.fr.data.core.db.dialect.AbstractDialect;
import com.fr.data.impl.Connection;
import com.fr.plugin.ExtraClassManager;
import com.fr.stable.fun.FunctionHelper;
import com.fr.stable.fun.FunctionProcessor;
import com.fr.stable.fun.impl.AbstractFunctionProcessor;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * adsmysql, odbc.
 * metadata, sql.
 * , odbc, , dialect.
 */
public class AdsMysqlOdbcDialect extends AbstractDialect {

    public static final String PLUGIN_ID =
"com.fr.plugin.adsDialect.DialectCreatorImpl";
    private static final FunctionProcessor FUNCTION_RECORD = new
AbstractFunctionProcessor() {
        @Override
        public int getId() {
            return FunctionHelper.generateFunctionID(PLUGIN_ID);
        }

        @Override
        public String getLocaleKey() {
            return "FR-Plugin_Ads-Dialect";
        }
```

```
    };


    /**
     * sql
     *
     * @param query sql
     * @return sql
     */
    public String createSQL4Columns(String query) {
        return query;
    }

    /**
     *
     *
     * @return
     */
    public TableProcedure[] getAllTableProcedure(Connection database,
String type) {
        FunctionProcessor processor =
ExtraClassManager.getInstance().getFunctionProcessor();
        if (processor != null) {
            processor.recordFunction(FUNCTION_RECORD);
        }

        java.sql.Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try {
            List sqlTableList = new ArrayList();
            connection = database.createConnection();

            statement = connection.createStatement();
            resultSet = statement.executeQuery("SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES");
            while (resultSet.next()) {
                sqlTableList.add(new TableProcedure(null,
resultSet.getString(1), type, this));
            }
            return (TableProcedure[]) sqlTableList.toArray(new
TableProcedure[sqlTableList.size()]);
        } catch (Exception e) {
            FRContext.getLogger().error(e.getMessage());
        } finally {
            release(connection, statement, resultSet);
        }

        return new TableProcedure[0];
    }

    private void release(java.sql.Connection connection, Statement
statement, ResultSet resultSet) {
```

```
        try {
            if (resultSet != null) {
                resultSet.close();
            }

            if (statement != null) {
                statement.close();
            }

            if (connection != null) {
                connection.close();
            }
        } catch (Exception e) {

        }
    }

    public List getTableFieldsInfor(java.sql.Connection conn, String
tableName, String schema, String dbLink){
        String query = "select * from " + (schema == null ? "" : (schema +
".")) + tableName;
        List result = new ArrayList();

        try {
            ColumnInformation[] informations =
DBUtils.checkInColumnInformation(conn, this, query);
            for(ColumnInformation information : informations){
                Map field = new HashMap();
                field.put("column_name", information.getColumnName());
                field.put("column_comment", "");
                field.put("column_type", information.getColumnType());
                field.put("column_size", information.getColumnSize());
                field.put("column_key", false);
                result.add(field);
            }

        } catch (SQLException e) {
            FRContext.getLogger().error(e.getMessage());
        }

        return result;
```

```
        }
    }
```

从上面两个接口的实现类就能够很容易明白如何处理数据库差异，自定义数据库方言来实现对某些小众数据库的支持。

## 源码

如果你需要更清晰的了解这个接口的使用，可以参照在自定义OdbcDialect

的实现：http://cloud.finedevelop.com:2015/projects/BA/repos/plugins-free/browse/plugin-ads_odbc_dialect