**DREAM 简介**

**DREAM是一款持久层框架，从设计结构到使用方式与MyBatis大同小异，DREAM有自己的语法标准，目前可以说，只支持MYSQL语法，因此在非MYSQL环境下，特有的SQL语句将不会识别，但DREAM不气馁，DREAM对常用的函数预写好了翻译，未翻译的函数提供了可翻译的方案，DREAM本质就是一款基于翻译的持久层框架。细味品尝DREAM，会让你内心一惊的。**

**开始DREAM教程**

**DREAM配置类Configuration**

public class Configuration {  
 private MapperFactory mapperFactory;  
 private TableFactory tableFactory;  
 private CacheFactory cacheFactory;  
 private TypeHandlerFactory typeHandlerFactory;  
 private DialectFactory dialectFactory;  
 private PluginFactory pluginFactory;  
 private ListenerFactory listenerFactory;  
 private TransactionFactory transactionFactory;  
 private DataSourceFactory dataSourceFactory;

**配置类属性说明**

|  |  |
| --- | --- |
| **接口** | **简述** |
| **MapperFactory** | **接口代理，配置SQL信息** |
| **TableFactory** | **类属性与数据库绑定，映射** |
| **CacheFactory** | **配置缓存** |
| **TypeHandlerFactory** | **类型转换器** |
| **DialectFactory** | **SQL方言** |
| **PluginFactory** | **配置插件** |
| **ListenerFactory** | **监听器** |
| **TransactionFactory** | **配置事务** |
| **DataSourceFactory** | **配置数据源** |

**DREAM是由以上工厂组建而成，熟悉了工厂的设计，也就熟悉了DREAM原理，开发者可以改写上述任何一种接口，从根本改变DREAM，因此，DREAM的灵活度非常之高。**

**DREAM可以由XML文件构建，也就是声明工厂，具体看demo。**

**MapperFactory类**

package com.moxa.dream.module.mapper.factory;  
public interface MapperFactory {  
 void addMapper(Configuration configuration, Class type);  
 <T> T getMapper(Class<T> type, MapperHandler mapperHandler);  
 Collection<Class> getMapperTypeList();  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **addMapper** | **默认实现，若接口type存在Mapper注解，扫描类方法，若方法存在Sql注解，SQL语句与类方法绑定组建成对象MethodInfo，如若Mapper指向的XML SQL配置文件存在，则优先XML配置** |
| **getMapper** | **默认实现，返回接口对象的代理类** |
| **getMapperTypeList** | **返回总共代理的接口类，为整合springboot服务** |

**Mapper注解类，属性value,指向xml配置文件**

package com.moxa.dream.module.annotation;  
public @interface Mapper {  
 String value() default "";  
}

**SQL注解类 value:配置的SQL语句，timeOut:查询超时时间**

package com.moxa.dream.module.annotation;  
public @interface Sql {  
 String value();  
 String timeOut() default "";  
}

**Result：注解**

package com.moxa.dream.module.annotation;  
public @interface Result {  
 Class<? extends Collection> rowType() default NullObject.class;  
 Class<?> colType() default NullObject.class;  
 boolean generatedKeys() default false;  
}

|  |  |
| --- | --- |
| **方法** | **简述** |
| **rowType** | **方法返回值集合类型，NullObject表示自动推断，如果只返回单对象，可以指定NonCollection** |
| **colType** | **方法返回值集合元素类型，NullObject表示自动推断** |
| **generatedKeys** | **如果是插入操作，返回自增主键值** |

**TableFactory类**

package com.moxa.dream.module.table.factory;  
public interface TableFactory {  
 void addTableInfo(Class type);  
 TableInfo getTableInfo(String table);  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **addTableInfo** | **默认实现，若类type存在Table注解，绑定类与数据库之间关系** |
| **getTableInfo** | **根据表名(忽略大小写)获取TableInfo** |

**CacheFactory类**

package com.moxa.dream.module.cache.factory;  
public interface CacheFactory {  
 void setProperties(Properties properties);  
 Cache getCache();  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **setProperties** | **传入的Properties** |
| **getCache** | **返回的Cache对象** |

**Cache类**

package com.moxa.dream.module.cache;

public interface Cache {  
 void put(MappedStatement mappedStatement, Object value);  
 Object get(MappedStatement mappedStatement);  
 void remove(MappedStatement mappedStatement);  
 void clear();  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **put** | **缓存数据** |
| **get** | **获取缓存** |
| **remove** | **删除缓存** |
| **clear** | **清空缓存** |

**MappedStatement贯穿DREAM始终。**

**默认实现的两种缓存，缓存可以保证如若缓存有数据，则数据一定和数据库数据一致**

package com.moxa.dream.module.cache;  
public class DefaultCache implements Cache {

}

package com.moxa.dream.driver.cache;  
public class ThirdCache implements com.moxa.dream.module.cache.Cache {

}

**TypeHandlerFactory类**

package com.moxa.dream.module.typehandler.factory;  
public interface TypeHandlerFactory {  
 void wrapper(List<? extends TypeHandlerWrapper> typeHandlerWrapperList);  
 TypeHandler getTypeHandler(Class javaType, int jdbcType);  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **wrapper** | **解析TypeHandlerWrapper获取TypeHandler** |
| **getTypeHandler** | **根据java类与数据库类型获取对应的TypeHandler** |

**TypeHandlerWrapper类**

package com.moxa.dream.module.typehandler.wrapper;  
public interface TypeHandlerWrapper {  
 TypeHandler getTypeHandler();  
 Integer[] typeCode();  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **getTypeHandler** | **返回类型转化器** |
| **typeCode** | **类型转换器适用的java类与数据库类型，typeCode：java类与数据库类型的哈希值** |

**DialectFactory类**

package com.moxa.dream.module.dialect;  
public interface DialectFactory {  
 default void setProperties(Properties properties) {  
 }  
 void setDialect(ToSQL toSQL);  
 PackageStatement compile(MethodInfo methodInfo);  
 MappedStatement compile(MethodInfo methodInfo, Object arg);  
 CacheKey getCacheKey(MethodInfo methodInfo);  
 void wrapper(PackageStatement statement, MethodInfo methodInfo);

}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **setProperties** | **传入的Properties对象** |
| **setDialect** | **传入的翻译器** |
| **compile(MethodInfo)** | **将MethodInfo转化抽象树** |
| **compile(MethodInfo,Object)** | **将MethodInfo和Object编译成MappedStatement** |
| **getCacheKey** | **返回MethodInfo的唯一key** |
| **wrapper** | **抽象树自定义包裹处理** |

**PluginFactory类**

package com.moxa.dream.module.plugin.factory;  
public interface PluginFactory {  
 void interceptor(Interceptor[] interceptors);  
 Object plugin(Object target);  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **interceptor** | **传入的拦截器** |
| **plugin** | **插件的对象** |

**默认实现的插件工厂**

package com.moxa.dream.module.plugin.factory;  
public class JavaPluginFactory extends AbstractPluginFactory {

}

**ListenerFactory类**

package com.moxa.dream.module.core.listener.factory;  
public interface ListenerFactory {  
 void listener(Listener[] listeners);  
 QueryListener[] getQueryListener();  
 UpdateListener[] getUpdateListener();  
 InsertListener[] getInsertListener();  
 DeleteListener[] getDeleteListener();  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **listener** | **传入的监听器** |
| **getQueryListener** | **返回负责查询的监听器** |
| **getUpdateListener** | **返回负责更新的监听器** |
| **getInsertListener** | **返回负责插入的监听器** |
| **getDeleteListener** | **返回负责删除的监听器** |

**四种监听器统一继承Listener**

package com.moxa.dream.module.core.listener;  
public interface Listener {  
 void before(MappedStatement mappedStatement);  
 void afterReturn(Object result, MappedStatement mappedStatement);  
 void exception(Exception e, MappedStatement mappedStatement);  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **before** | **执行操作之前操作** |
| **after** | **成功之后操作** |
| **exception** | **异常后操作** |

**TransactionFactory类**

package com.moxa.dream.module.transaction.factory;  
public interface TransactionFactory {  
 default void setProperties(Properties properties) {  
 }  
 Transaction getTransaction();  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **setProperties** | **传入的properties** |
| **getTransaction** | **返回的事务** |

**Transaction类**

package com.moxa.dream.module.transaction;  
public interface Transaction {  
 void setDataSource(DataSource dataSource);  
 Connection getConnection() throws SQLException;  
 void commit();  
 void rollback();  
 void close();  
 boolean isAutoCommit();  
 void setAutoCommit(boolean autoCommit);  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **setDataSource** | **传入的DataSource** |
| **getConnection** | **获取Connection连接** |
| **commit** | **提交操作** |
| **rollback** | **回滚操作** |
| **close** | **关闭操作** |
| **isAutoCommit** | **判断是否自动提交** |
| **setAutoCommit** | **设置提交方式** |

**DataSourceFactory类**

package com.moxa.dream.module.datasource;  
public interface DataSourceFactory {  
 void setProperties(Properties properties);  
 DataSource getDataSource();  
}

|  |  |
| --- | --- |
| **方法名** | **简述** |
| **setProperties** | **传入的properties** |
| **getDataSource** | **获取DataSource** |

**整合SpringBoot**

**引入pom**

<dependency>  
 <groupId>com.moxa</groupId>  
 <artifactId>dream-boot-starter</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
</dependency>

**接口Mapper注解 以及SQL注解**

@Mapper  
public interface CityMapper {  
 @Sql("select id, name, state, country from city where state = @$(state)")  
 City findByState(String state);  
 @Sql("update city set name=null where state = @$(state)")  
 void updateCity(String state);  
}

**类与数据库表绑定注解Table，类属性与表字段绑定注解Column,声明主键注解Id**

@Table  
public class City implements Serializable {  
 @Id  
 @Column  
 private Integer id;  
 @Column

private String name;

**application.properties配置**

dream.config-location=config.xml  
dream.dialect=com.moxa.dream.antlr.sql.ToMYSQL