# 动态数据源

[Spring 实现数据库读写分离](https://www.cnblogs.com/surge/p/3582248.html)

## 项目中应用：

Flight-base项目中：

1. dao层配置数据源：

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns="http://www.springframework.org/schema/beans"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="abstractDataSource" abstract="true" class="com.alibaba.druid.pool.DruidDataSource" init-method="init"  
 destroy-method="close">  
 <property name="maxWait" value="60000"/>  
 <property name="minIdle" value="1"/>  
 <property name="timeBetweenEvictionRunsMillis" value="3000"/>  
 <property name="minEvictableIdleTimeMillis" value="300000"/>  
 <property name="validationQuery" value="SELECT 'x'"/>  
 <property name="testWhileIdle" value="true"/>  
 <property name="testOnBorrow" value="false"/>  
 <property name="testOnReturn" value="false"/>  
 </bean>  
  
 <bean id="dataSourceWrite" parent="abstractDataSource">  
 <property name="url" value="${flight.base.w.jdbc.jdbcUrl}"/>  
 <property name="username" value="${flight.base.w.jdbc.username}"/>  
 <property name="password" value="${flight.base.w.jdbc.password}"/>  
 <!-- 配置初始化大小、最小、最大 -->  
 <property name="initialSize" value="${flight.base.w.jdbc.minConnections}"/>  
 <property name="minIdle" value="5"/>  
 <property name="maxActive" value="${flight.base.w.jdbc.maxConnections}"/>  
 </bean>  
  
 <bean id="dataSourceRead" parent="abstractDataSource">  
 <property name="url" value="${flight.base.r.jdbc.jdbcUrl}"/>  
 <property name="username" value="${flight.base.r.jdbc.username}"/>  
 <property name="password" value="${flight.base.r.jdbc.password}"/>  
 <!-- 配置初始化大小、最小、最大 -->  
 <property name="initialSize" value="${flight.base.r.jdbc.minConnections}"/>  
 <property name="minIdle" value="5"/>  
 <property name="maxActive" value="${flight.base.r.jdbc.maxConnections}"/>  
 </bean>  
  
 <bean id="dataSource" class="com.jd.airplane.infra.mybatis.datasource.DynamicDataSource" lazy-init="true">  
 <property name="writeDataSource" ref="dataSourceWrite"/>  
 <property name="readDataSource" ref="dataSourceRead"/>  
 <!-- 设置默认的目标数据源 -->  
 <property name="defaultTargetDataSource" ref="dataSourceWrite"/>  
 <property name="targetDataSources">  
 <map key-type="java.lang.String" value-type="javax.sql.DataSource">  
 <entry key="WRITE" value-ref="dataSourceWrite"/>  
 <entry key="READ" value-ref="dataSourceRead"/>  
 </map>  
 </property>  
 </bean>  
  
 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  
 <property name="dataSource" ref="dataSource"/>  
 <property name="configLocation" value="classpath:mybatis/mybatis-config.xml"/>  
 <property name="mapperLocations" value="classpath:mybatis/mapper/com/jd/airplane/flight/base/model/\*/\*.xml"/>  
 </bean>  
  
 <bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">  
 <constructor-arg index="0" ref="sqlSessionFactory"/>  
 <constructor-arg index="1" value="REUSE"/>  
 </bean>  
</beans>

## 枚举字段与数据表字段的映射、

[**MyBatis里字段到枚举类型的转换/映射**](https://www.cnblogs.com/jeffen/p/6380982.html)

项目中使用：

*/\*\*  
 \* 用于枚举类型和数据库的自动映射  
 \* EnumTypeHandler : 使用枚举字符串名称传递参数  
 \* Created by yangyanping on 2018/3/4.  
 \*/*public class EnumCodeTypeHandler <E extends Enum<E> & IEnumCode<E>> extends EnumTypeHandler<E> {  
 private Class<E> type;  
 public EnumCodeTypeHandler(Class<E> type) {  
 super(type);  
 this.type = type;  
 }  
 @Override  
 public void setNonNullParameter(PreparedStatement preparedStatement, int i, E parameter, JdbcType jdbcType) throws SQLException {  
 preparedStatement.setInt(i, parameter.getCode());  
 }  
 @Override  
 public E getNullableResult(ResultSet resultSet, String columnName) throws SQLException {  
 //使用列名获取数据  
 Integer code = resultSet.getInt(columnName);  
  
 return parse(code);  
 }  
 @Override  
 public E getNullableResult(ResultSet resultSet, int columnIndex) throws SQLException {  
 //使用列下标获取数据  
 Integer code = resultSet.getInt(columnIndex);  
  
 return parse(code);  
 }  
 @Override  
 public E getNullableResult(CallableStatement callableStatement, int columnIndex) throws SQLException {  
 Integer code = callableStatement.getInt(columnIndex);  
  
 return parse(code);  
 }  
 private E parse(Integer code) {  
 if (null == code) {  
 return null;  
 }  
 for (E e : type.getEnumConstants()) {  
 if (code.equals(e.getCode())) {  
 return e;  
 }  
 }  
 return null;  
 }  
}

具体枚举类使用上面handler

public enum UserLevelEnum implements IEnumCode<UserLevelEnum> {

public interface IEnumCode<E extends Enum<E>> {  
 int getCode();  
}

public class UserLevelEnumCodeHandler extends EnumCodeTypeHandler<UserLevelEnum> {  
 public UserLevelEnumCodeHandler() {  
 super(UserLevelEnum.class);  
 }  
}

## <choose>与<if>

[MyBatis-动态SQL的if、choose、when、otherwise、trim、where、set、foreach使用](https://blog.csdn.net/qq_32588349/article/details/51541871)

[mybatis】IF判断的坑](http://cheng-xinwei.iteye.com/blog/2008200)

[MyBatis中的OGNL教程](https://blog.csdn.net/isea533/article/details/50061705)

有时候我们并不想应用所有的条件，而只是想从多个选项中选择一个。而使用if标签时，只要test中的表达式为 true，就会执行 if 标签中的条件。MyBatis 提供了 choose 元素。if标签是与(and)的关系，而 choose 是或(or)的关系。

choose标签是按顺序判断其内部when标签中的test条件出否成立，如果有一个成立，则 choose 结束。当 choose 中所有 when 的条件都不满则时，则执行 otherwise 中的sql。类似于Java 的 switch 语句，choose 为 switch，when 为 case，otherwise 则为 default。

Mybatis中等于是==或eq

项目中使用

<choose>  
 <when test="inValidTime == null or inValidTime == 0">  
 <!--查有效活动-->  
 <if test="endTime != null and endTime != ''">  
 AND END\_TIME &gt;= #{endTime}  
 </if>  
 </when>  
 <when test="inValidTime != null and inValidTime &gt; 0">  
 <!--查无效活动-->  
 <if test="endTime != null and endTime != ''">  
 AND END\_TIME &gt;= #{endTime}-#{inValidTime}\*86400000 AND END\_TIME &lt;= #{endTime}  
 </if>  
 <if test="statusEnum != null and statusEnum != '' and endTime != null and endTime != ''">  
 OR (ACT\_STATUS = #{statusEnum,typeHandler=com.jd.airplane.base.dao.handler.ActivityStatusEnumCodeHandler}  
 AND unix\_timestamp(MODIFIED)\*1000 &gt;= #{endTime}-#{inValidTime}\*86400000 AND unix\_timestamp(MODIFIED)\*1000 &lt;= #{endTime})  
 </if>  
 </when>  
 <otherwise>  
 <!--不满足前两个条件则查全部活动-->  
 <if test="beginTime != null and beginTime != ''">  
 AND BEGIN\_TIME = #{beginTime}  
 </if>  
 <if test="endTime != null and endTime != ''">  
 AND END\_TIME = #{endTime}  
 </if>  
 </otherwise>  
</choose>

## #与$符号区别

1. 介绍  
    mybatis 中使用 Mapper.xml里面的配置进行 sql 查询，经常需要动态传递参数，例如我们需要根据用户的姓名来筛选用户时，sql 如下：  
   select \* from user where name = "Jack";    
   上述 sql 中，我们希望 name 后的参数 "Jack" 是动态可变的，即不同的时刻根据不同的姓名来查询用户。在 Mapper.xml文件中使用如下的 sql 可以实现动态传递参数 name：  
   select \* from user where name = #{name};    
   或者是：  
   select \* from user where name = ${name};    
   二、$ 与 #  
   1. 区别 :  
   动态 SQL 是 mybatis 的强大特性之一，也是它优于其他 ORM 框架的一个重要原因。mybatis 在对 sql 语句进行预编译之前，会对 sql 进行动态解析，解析为一个 BoundSql 对象，也是在此处对动态 SQL 进行处理的。在动态 SQL 解析阶段， #{ } 和 ${ } 会有不同的表现。  
   #{ }：解析为一个 JDBC 预编译语句（prepared statement）的参数标记符。  
   例如，Mapper.xml中如下的 sql 语句：  
   select \* from user where name = #{name};    
   动态解析为：  
   select \* from user where name = ?;    
   一个 #{ } 被解析为一个参数占位符 ? 。  
   而${ } 仅仅为一个纯碎的 string 替换，在动态 SQL 解析阶段将会进行变量替换。  
   例如，Mapper.xml中如下的 sql：  
   select \* from user where name = ${name};    
   当我们传递的参数为 "Jack" 时，上述 sql 的解析为：  
   select \* from user where name = "Jack";    
   预编译之前的 SQL 语句已经不包含变量了，完全已经是常量数据了。  
   综上所得， ${ } 变量的替换阶段是在动态 SQL 解析阶段，而 #{ }变量的替换是在 DBMS 中。  
   三、用法  
   1、能使用 #{ } 的地方就用 #{ }  
   首先这是为了性能考虑的，相同的预编译 sql 可以重复利用。其次，${ } 在预编译之前已经被变量替换了，这会存在 sql 注入问题。例如，如下的 sql：  
   select \* from ${tableName} where name = #{name}    
   假如，我们的参数 tableName 为 user; delete user; --，那么 SQL 动态解析阶段之后，预编译之前的 sql 将变为：  
   select \* from user; delete user; -- where name = ?;    
   -- 之后的语句将作为注释，不起作用，因此本来的一条查询语句偷偷的包含了一个删除表数据的 SQL。  
   2. 表名作为变量时，必须使用 ${ }  
   这是因为，表名是字符串，使用 sql 占位符替换字符串时会带上单引号 ''，这会导致 sql 语法错误，例如：  
   select \* from #{tableName} where name = #{name};    
   预编译之后的sql 变为：  
   select \* from ? where name = ?;    
   假设我们传入的参数为 tableName = "user" , name = "Jack"，那么在占位符进行变量替换后，sql 语句变为：  
   select \* from 'user' where name='Jack';    
   上述 sql 语句是存在语法错误的，表名不能加单引号 ''（注意，反引号 ``是可以的）。  
   四、sql预编译  
   1. 定义：  
   sql 预编译指的是数据库驱动在发送 sql 语句和参数给 DBMS 之前对 sql 语句进行编译，这样 DBMS 执行 sql 时，就不需要重新编译。  
   2. 为什么需要预编译  
   JDBC 中使用对象 PreparedStatement 来抽象预编译语句，使用预编译。预编译阶段可以优化 sql 的执行。预编译之后的 sql 多数情况下可以直接执行，DBMS 不需要再次编译，越复杂的sql，编译的复杂度将越大，预编译阶段可以合并多次操作为一个操作。预编译语句对象可以重复利用。把一个 sql 预编译后产生的 PreparedStatement 对象缓存下来，下次对于同一个sql，可以直接使用这个缓存的 PreparedState 对象。mybatis 默认情况下，将对所有的 sql 进行预编译。

## 无参构造函数

因为mybatis中，查询数据表时，会生成对象的cglib代理（通过.class字节码生成子类），所以需要无参构造函数

## parameterType

selectList方法：

public <E> List<E> selectList(String statement, Object parameter, RowBounds rowBounds) {  
 return this.sqlSessionProxy.selectList(statement, parameter, rowBounds);  
}

参数：parameter类型是Object，可以是map，对象，list，数组，及基本类型

传入参数与，mapper.xml文件中的使用关系见下面链接

可以传递一个List或Array类型的对象作为参数,MyBatis会自动的将List或Array对象包装到一个Map对象中,List类型对象会使用list作为键名,而Array对象会用array作为键名。

[MyBatis传入参数与parameterType](https://www.cnblogs.com/ruiati/p/6410339.html)

Foreach属性

| **属性** | **描述** |
| --- | --- |
| item | 循环体中的具体对象。支持属性的点路径访问，如item.age,item.info.details。 具体说明：在list和数组中是其中的对象，在map中是value。 该参数为必选。 |
| collection | 要做foreach的对象，作为入参时，List<?>对象默认用list代替作为键，数组对象有array代替作为键，Map对象没有默认的键。 当然在作为入参时可以使用@Param("keyName")来设置键，设置keyName后，list,array将会失效。 除了入参这种情况外，还有一种作为参数对象的某个字段的时候。举个例子： 如果User有属性List ids。入参是User对象，那么这个collection = "ids" 如果User有属性Ids ids;其中Ids是个对象，Ids有个属性List id;入参是User对象，那么collection = "ids.id" 上面只是举例，具体collection等于什么，就看你想对那个元素做循环。 该参数为必选。 |
| separator | 元素之间的分隔符，例如在in()的时候，separator=","会自动在元素中间用“,“隔开，避免手动输入逗号导致sql错误，如in(1,2,)这样。该参数可选。 |
| open | foreach代码的开始符号，一般是(和close=")"合用。常用在in(),values()时。该参数可选。 |
| close | foreach代码的关闭符号，一般是)和open="("合用。常用在in(),values()时。该参数可选。 |
| index | 在list和数组中,index是元素的序号，在map中，index是元素的key，该参数可选。 |

### 项目中使用：

Java调用：参数是

Map<String, DictEnum> parameter = Collections.*singletonMap*("dict", dict);

而DictEnum中的属性

public enum DictEnum {  
 //机场  
 *AIRPORT*("dict\_airport", new HashMap<String, String>() {  
 {  
 put(*CODE*, "code");  
 put(*CHINESE\_NAME*, "name");  
 put(*CITY\_CODE*, "city\_code");  
 put(*CITY\_NAME*, "city\_name");  
 put(*PINYIN*, "spell\_name");  
 put(*SHORT\_PINYIN*, "simple\_spell\_name");  
 }  
 })

DictEnum(String table, Map<String, String> fieldMap) {  
 this.table = table;  
 this.fieldMap = Collections.*unmodifiableMap*(fieldMap);  
}

调用

list = dictSqlSessionTemplate.selectList(*SELECT\_ITEM\_LIST*, parameter, new RowBounds(offset, limit));

mapper.xml中collection="dict.fieldMap.keySet()"也可以

<select id="selectItemList" parameterType="map" resultType="java.util.Map">  
 SELECT id  
 <foreach collection="dict.fieldMap.keys" item="key">  
 ,${dict.fieldMap[key]} as ${key}  
 </foreach>  
 FROM ${dict.table}  
 WHERE  
 is\_delete = 0  
 ORDER BY id  
</select>

