(9条消息)MyBatis直接执行SQL的工具SqlMapper_Java_MarkArch的博客-CSDN博客

笔记本:	A1-Tech
创建时间:	2020/4/27 19:40
URL:	https://blog.csdn.net/markarch/article/details/51150324?utm_source=distribute.pc_relevant.none-task-blog-baidujs-4

MyBatis直接执行SQL的工具SqlMapper

转载 MarkArch 最后发布于2016-04-14 11:41:32 阅读数 493 收藏

目录[-]

可能有些人也有过类似需求，一般都会选择使用其他的方式如Spring-JDBC等方式解决。

能否通过MyBatis实现这样的功能呢?

为了让通用Mapper更彻底的支持多表操作以及更灵活的操作，在2.2.0版本增加了一个可以直接执行SQL的新类`SqlMapper`。

注：3.3.0版本去掉了这个类，这个类现在在EntityMapper项目

通过这篇博客，我们来了解一下`SqlMapper`。

SqlMapper提供的方法

`SqlMapper`提供了以下这些公共方法:

- `Map<String,Object> selectOne(String sql)`
- `Map<String,Object> selectOne(String sql, Object value)`
- `<T> T selectOne(String sql, Class<T> resultType)`
- `<T> T selectOne(String sql, Object value, Class<T> resultType)`
- `List<Map<String,Object>> selectList(String sql)`
- `List<Map<String,Object>> selectList(String sql, Object value)`
- `<T> List<T> selectList(String sql, Class<T> resultType)`
- `<T> List<T> selectList(String sql, Object value, Class<T> resultType)`
- `int insert(String sql)`
- `int insert(String sql, Object value)`
- `int update(String sql)`
- `int update(String sql, Object value)`
- `int delete(String sql)`
-

```
int delete(String sql, Object value)
```

一共14个方法，这些方法的命名和参数和SqlSession接口的很像，只是基本上第一个参数都成了sql。

其中Object value为入参，入参形式和SqlSession中的入参一样，带有入参的方法，在使用时sql可以包含#{param}或${param}形式
入参来传值。需要的参数过多的时候，参数可以使用Map类型。另外这种情况下的sql还支持下面这种复杂形式：

```
String sql = "<script>select * from sys_user where 1=1"  +
        "<if test=\"usertype != null\">usertype = #{usertype}</if></script>";

```

这种情况用的比较少，不多说。

不带有Object value的所有方法，sql中如果有参数需要手动拼接成一个可以直接执行的sql语句。

在selectXXX方法中，使用Class<T> resultType可以指定返回类型，否则就是Map<String,Object>类型。

实例化SqlMapper

SqlMapper构造参数public SqlMapper(SqlSession sqlSession)，需要一个入参SqlSession sqlSession，在一般系统中，可以

```
SqlSession sqlSession = (...);//通过某些方法获取sqlSession
//创建sqlMapper
SqlMapper sqlMapper = new SqlMapper(sqlSession);
```

如果使用的Spring，那么可以按照下面的方式配置<bean>:

```
<bean id="sqlMapper" class="com.github.abel533.sql.SqlMapper" scope="prototype">
  <constructor-arg ref="sqlSession"/>
</bean>
```

在Service中使用的时候可以直接使用@Autowired注入。

简单例子

在src/test/java目录的com.github.abel533.sql包中包含这些方法的测试。

下面挑几个看看如何使用。

selectList

```
//查询，返回List<Map>
List<Map<String, Object>> list = sqlMapper.selectList("select * from country where id < 11");

//查询，返回指定的实体类
List<Country> countryList = sqlMapper.selectList("select * from country where id < 11", Country.class);

//查询，带参数
countryList = sqlMapper.selectList("select * from country where id < #{id}", 11, Country.class);

//复杂点的查询，这里参数和上面不同的地方，在于传入了一个对象
Country country = new Country();
country.setId(11);
countryList = sqlMapper.selectList("<script>" +
        "select * from country " +
        "   <where>" +
        "       <if test=\"id != null\">" +
        "           id &lt; #{id}" +
        "       </if>" +
        "   </where>" +
        "</script>", country, Country.class);
```

selectOne

```
Map<String, Object> map = sqlMapper.selectOne("select * from country where id = 35");

map = sqlMapper.selectOne("select * from country where id = #{id}", 35);

Country country = sqlMapper.selectOne("select * from country where id = 35", Country.class);

country = sqlMapper.selectOne("select * from country where id = #{id}", 35, Country.class);
```

insert,update,delete

```
//insert
int result = sqlMapper.insert("insert into country values(1921,'天朝','TC')");

Country tc = new Country();
tc.setId(1921);
tc.setCountryname("天朝");
tc.setCountrycode("TC");
//注意这里的countrycode和countryname故意写反的
result = sqlMapper.insert("insert into country values(#{id},#{countrycode},#{countryname})"
                          , tc);


//update
result = sqlMapper.update("update country set countryname = '天朝' where id = 35");

tc = new Country();
tc.setId(35);
tc.setCountryname("天朝");

int result = sqlMapper.update("update country set countryname = #{countryname}" +
           " where id in(select id from country where countryname like 'A%')", tc);


//delete
result = sqlMapper.delete("delete from country where id = 35");
result = sqlMapper.delete("delete from country where id = #{id}", 35);
```

注意

通过上面这些例子应该能对此有个基本的了解，但是如果你使用参数方式，建议阅读下面的文章：

> 深入了解MyBatis参数

实现原理

2015-03-09：最初想要设计这个功能的时候，感觉会很复杂，想的也复杂，需要很多个类，因此当时没有实现。

2015-03-10：突发奇想，设计了现在的这种方式。并且有种强烈的感觉就是幸好昨天没有尝试去实现，因为昨天晚上思考这个问题的时（10号）是晚上7点开始思考。我很庆幸在一个更清醒的状态下去写这段代码。

下面简单说思路和实现方式。

在写MyBatis分页插件的时候熟悉了MappedStatement类。

在写通用Mapper的时候熟悉了xml转SqlNode结构。

如果我根据SQL动态的创建一个MappedStatement，然后使用MappedStatement的id在sqlSession中执行不就可以了吗？

想到这一点，一切就简单了。

看看下面select查询创建MappedStatement的代码：

```
/**
 * 创建一个查询的MS
```

```
 * 4    * @param msId
 * @param sqlSource 执行的sqlSource
 * @param resultType 返回的结果类型
 */
private void newSelectMappedStatement(String msId, SqlSource sqlSource, final Class<?> resultType) {
    MappedStatement ms = new MappedStatement.Builder(
            configuration, msId, sqlSource, SqlCommandType.SELECT)
        .resultMaps(new ArrayList<ResultMap>() {
            {
                add(new ResultMap.Builder(configuration,
                        "defaultResultMap",
                        resultType,
                        new ArrayList<ResultMapping>(0)).build());
            }
        })
        .build();
    //缓存
    configuration.addMappedStatement(ms);
}
```

代码是不是很简单，这段代码的关键是参数sqlSource，下面是创建SqlSource的方法，分为两种。

一种是一个完整的sql，不需要参数的，可以直接执行的：

```
StaticSqlSource sqlSource = new StaticSqlSource(configuration, sql);
```

其中configuration从sqlSession中获取，sql就是用户传入到sql语句，是不是也很简单？

另一种是支持动态sql的，支持参数的SqlSource：

```
SqlSource sqlSource = languageDriver.createSqlSource(configuration, sql, parameterType);

```

是不是也很简单？这个方法其实可以兼容上面的StaticSqlSource，这里比上面多了一个parameterType，因为这儿是可以传递参数的configuration中获取的。

是不是很简单？

我一开始也没想到MyBatis直接执行sql实现起来会这么的容易。

insert,delete,update方法的创建更容易，因为他们的返回值都是int，所以处理起来更简单，有兴趣的可以去通用Mapper下的包com.看SqlMapper的源码。