15 | mBatis：如何将SQL语句配置化?

2023-04-14 郭屹 来自北京

《手把手带你写一个MiniSpring》

你好，我是郭屹。今天我们继续手写 MiniSpring。这节课我们要模仿 MyBatis，将 SQL 语句配置化。

上一节课，在已有的 JDBC Template 基础之上，我们按照专门的事情交给专门的部件来做的思路，对它做了进一步地拆解，抽取出了数据源 DataSource 这个部件，然后我们把 SQL 语句参数的处理独立成了一个 ArgumentPreparedStatementSetter，之后对于返回结果，我们提供了两个东西，一个 RowMapper 和一个 RowMapperResultSetExtractor，把一条数据库记录和一个记录集转换成对象和对象列表，便利了上层应用程序。最后为了提高性能，我们还引入了一个简单的数据库连接池。

现在执行的 SQL 语句本身还是硬编码在程序中的，所以这节课，我们就模仿 MyBatis，把 SQL 语句放到程序外面的配置文件中。

MyBatis 简介

我们先来简单了解一下 MyBatis。

> 官方说法：MyBatis is a first class persistence framework with support for custom SQL, stored procedures and advanced mappings. MyBatis eliminates almost all of the JDBC code and manual setting of parameters and retrieval of results. MyBatis can use simple XML or Annotations for configuration and map primitives, Map interfaces and Java POJOs (Plain Old Java Objects) to database records.

从官方的资料里我们知道，MyBatis 的目标是构建一个框架来支持自定义 SQL、存储过程和复杂的映射，它将手工的 JDBC 代码都简化掉了，通过配置完成数据库记录与 Java 对象的转换。当然，MyBatis 不只是把 SQL 语句写到外部配置文件这么简单，它还干了好多别的工作，比如 ORM、缓存等等，我们这里只讨论 SQL 语句配置化。

在 MyBatis 的常规使用办法中，程序以这个 SqlSessionFactory 为中心，来创建一个 SqlSession，然后执行 SQL 语句。

你可以看一下简化后的代码。

复制代码

```
try (SqlSession session = sqlSessionFactory.openSession()) {
    Blog blog = session.selectOne(
            "org.mybatis.example.BlogMapper.selectBlog", 101);
}
```

上面代码的大意是先用 SqlSessionFactory 创建一个 SqlSession，然后把要执行的 SQL 语句的 id（org.mybatis.example.BlogMapper.selectBlog）和 SQL 参数（101），传进 session.selectOne() 方法，返回查询结果对象值 Blog。

凭直觉，这个 session 应当是包装了数据库连接信息，而这个 SQL id 应当是指向的某处定义的 SQL 语句，这样就能大体跟传统的 JDBC 代码对应上了。

我们就再往下钻研一下。先看这个 SqlSessionFactory 是怎么来的，一般在应用程序中这么写。

复制代码

```
String resource = "org/mybatis/example/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory =
          new SqlSessionFactoryBuilder().build(inputStream);
```

可以看出，它是通过一个配置文件由一个 SqlSessionFactoryBuider 工具来生成的，我们看看配置文件的简单示例。

复制代码

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "https://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

没有什么神奇的地方，果然就是数据库连接信息还有一些 mapper 文件的配置。用这些配置信息创建一个 Factory，这个 Factory 就知道该如何访问数据库了，至于具体执行的 SQL 语句，则是放在 mapper 文件中的，你可以看一下示例。

复制代码

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
```

```
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = #{id}
  </select>
</mapper>
```

我们看这个 mapper 文件，里面就是包含了 SQL 语句，给了 select 语句一个 namespace（org.mybatis.example.BlogMapper）以及 id（selectBlog），它们拼在一起就是上面程序中写的 SQL 语句的 sqlid（org.mybatis.example.BlogMapper.selectBlog）。我们还要注意这个 SQL 的参数占位符 #{id} 以及返回结果对象 Blog，它们的声明格式是 MyBatis 自己规定的。

转换成 JDBC 的语言，这里定义了这个 SQL 语句是一个 select 语句，命令是 select * from Blog where id = #{id}，参数是 id，返回结果对象对应的是 Blog，这条 SQL 语句有一个唯一的 sqlid 来代表。

现在我们几乎能想象出应用程序执行下面这行的时候在做什么了。

复制代码

```
    Blog blog = session.selectOne(
                    "org.mybatis.example.BlogMapper.selectBlog", 101);
```

一定就是用这个 id 去 mapper 文件里找对应的 SQL 语句，替换参数，然后执行，最后将数据库记录按照某种规则转成一个对象返回。整个过程跟我们在 JdbcTemplate 中做得很类似。

有了这个思路，我们就可以着手实现自己的 mBatis 了。

Mapper 配置

我们仿照 MyBatis，把 SQL 语句放在外部配置文件中。先在 resources 目录下建一个 mapper 目录，然后把 SQL 语句配置在这里，如 mapper/User_Mapper.xml 文件。

复制代码

```
  <?xml version="1.0" encoding="UTF-8"?>
```

```
<mapper namespace="com.test.entity.User">
    <select id="getUserInfo" parameterType="java.lang.Integer" resultType="com.
        select id, name,birthday
        from users
        where id=?
    </select>
</mapper>
```

这个配置中，也同样有基本的一些元素：SQL 类型、SQL 的 id、参数类型、返回结果类型、SQL 语句、条件参数等等。

自然，我们需要在内存中用一个结构来对应上面的配置，存放系统中的 SQL 语句的定义。

复制代码

```
package com.minis.batis;

public class MapperNode {
    String namespace;
    String id;
    String parameterType;
    String resultType;
    String sql;
    String parameter;

  public String getNamespace() {
    return namespace;
  }
  public void setNamespace(String namespace) {
    this.namespace = namespace;
  }
  public String getId() {
    return id;
  }
  public void setId(String id) {
    this.id = id;
  }
  public String getParameterType() {
    return parameterType;
  }
  public void setParameterType(String parameterType) {
    this.parameterType = parameterType;
  }
  public String getResultType() {
    return resultType;
```

```
  }
  public void setResultType(String resultType) {
    this.resultType = resultType;
  }
  public String getSql() {
    return sql;
  }
  public void setSql(String sql) {
    this.sql = sql;
  }
  public String getParameter() {
    return parameter;
  }
  public void setParameter(String parameter) {
    this.parameter = parameter;
  }
  public String toString(){
    return this.namespace+"."+this.id+" : " +this.sql;
  }
}
```

对它们的处理工作，我们仿照 MyBatis，用一个 SqlSessionFactory 来处理，并默认实现一个 DefaultSqlSessionFactory 来负责。

你可以看一下 SqlSessionFactory 接口定义。

复制代码

```
package com.minis.batis;

public interface SqlSessionFactory {
  SqlSession openSession();
  MapperNode getMapperNode(String name);
}
```

同时，我们仍然使用 IoC 来管理，将默认的 DefaultSqlSessionFactory 配置在 IoC 容器的 applicationContext.xml 文件里。

复制代码

```
    <bean id="sqlSessionFactory" class="com.minis.batis.DefaultSqlSessionFactory"
        <property type="String" name="mapperLocations" value="mapper"></property>
```

```
    </bean>
```

我们并没有再用一个 builder 来生成 Factory，这是为了简单一点。

这个 Bean，也就是这里配置的默认的 SqlSessionFactory，它在初始化过程中会扫描这个 mapper 目录。

复制代码

```
public void init() {
    scanLocation(this.mapperLocations);
}
```

而这个扫描跟以前的 Servlet 也是一样的，用递归的方式访问每一个文件。

复制代码

```
private void scanLocation(String location) {
    String sLocationPath = this.getClass().getClassLoader().getResource("").get
      File dir = new File(sLocationPath);
      for (File file : dir.listFiles()) {
          if(file.isDirectory()){ //递归扫描
            scanLocation(location+"/"+file.getName());
          }else{ //解析mapper文件
              buildMapperNodes(location+"/"+file.getName());
          }
      }
  }
```

最后对扫描到的每一个文件，要进行解析处理，把 SQL 定义写到内部注册表 Map 里。

复制代码

```
private Map<String, MapperNode> buildMapperNodes(String filePath) {
      SAXReader saxReader=new SAXReader();
      URL xmlPath=this.getClass().getClassLoader().getResource(filePath);

  Document document = saxReader.read(xmlPath);
  Element rootElement=document.getRootElement();

```

```
    String namespace = rootElement.attributeValue("namespace");

        Iterator<Element> nodes = rootElement.elementIterator();;
        while (nodes.hasNext()) { //对每一个sql语句进行解析
          Element node = nodes.next();
            String id = node.attributeValue("id");
            String parameterType = node.attributeValue("parameterType");
            String resultType = node.attributeValue("resultType");
            String sql = node.getText();

            MapperNode selectnode = new MapperNode();
            selectnode.setNamespace(namespace);
            selectnode.setId(id);
            selectnode.setParameterType(parameterType);
            selectnode.setResultType(resultType);
            selectnode.setSql(sql);
            selectnode.setParameter("");

            this.mapperNodeMap.put(namespace + "." + id, selectnode);
        }
      return this.mapperNodeMap;
  }
```

程序很简单，就是拿这个配置文件中的节点，读取节点的各项属性，然后设置到 MapperNode 结构中。注意，上面的解析可以看到最后这个完整的 id 是 namespace+ “.” +id，对应上面例子里的就是 com.test.entity.User.getUserInfo。还有，作为一个原理性示例，我们现在只能处理 select 这一种 SQL 语句，update 之类的语句留着之后扩展。考虑到有多种 SQL 命令，扩展的时候需要增加一个属性，表明这条 SQL 语句是读语句还是写语句。

你可以看一下 DefaultSqlSessionFactory 的完整代码。

复制代码

```
package com.minis.batis;

import java.io.File;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.sql.DataSource;
```

```
12 import org.dom4j.Document;
13 import org.dom4j.Element;
14 import org.dom4j.io.SAXReader;
15 import com.minis.beans.factory.annotation.Autowired;
16 import com.minis.jdbc.core.JdbcTemplate;
17
18 public class DefaultSqlSessionFactory implements SqlSessionFactory{
19   @Autowired
20   JdbcTemplate jdbcTemplate;
21
22   String mapperLocations;
23   public String getMapperLocations() {
24     return mapperLocations;
25   }
26   public void setMapperLocations(String mapperLocations) {
27     this.mapperLocations = mapperLocations;
28   }
29   Map<String,MapperNode> mapperNodeMap = new HashMap<>();
30   public Map<String, MapperNode> getMapperNodeMap() {
31     return mapperNodeMap;
32   }
33   public DefaultSqlSessionFactory() {
34   }
35
36   public void init() {
37       scanLocation(this.mapperLocations);
38   }
39     private void scanLocation(String location) {
40       String sLocationPath = this.getClass().getClassLoader().getResource("").get
41         File dir = new File(sLocationPath);
42         for (File file : dir.listFiles()) {
43             if(file.isDirectory()){
44               scanLocation(location+"/"+file.getName());
45             }else{
46                 buildMapperNodes(location+"/"+file.getName());
47             }
48         }
49     }
50
51   private Map<String, MapperNode> buildMapperNodes(String filePath) {
52     System.out.println(filePath);
53         SAXReader saxReader=new SAXReader();
54         URL xmlPath=this.getClass().getClassLoader().getResource(filePath);
55         try {
56       Document document = saxReader.read(xmlPath);
57       Element rootElement=document.getRootElement();
58
59       String namespace = rootElement.attributeValue("namespace");
60
```

```
61            Iterator<Element> nodes = rootElement.elementIterator();;
62            while (nodes.hasNext()) {
63              Element node = nodes.next();
64                String id = node.attributeValue("id");
65                String parameterType = node.attributeValue("parameterType");
66                String resultType = node.attributeValue("resultType");
67                String sql = node.getText();
68
69                MapperNode selectnode = new MapperNode();
70                selectnode.setNamespace(namespace);
71                selectnode.setId(id);
72                selectnode.setParameterType(parameterType);
73                selectnode.setResultType(resultType);
74                selectnode.setSql(sql);
75                selectnode.setParameter("");
76
77                this.mapperNodeMap.put(namespace + "." + id, selectnode);
78            }
79        } catch (Exception ex) {
80            ex.printStackTrace();
81        }
82        return this.mapperNodeMap;
83    }
84
85    public MapperNode getMapperNode(String name) {
86      return this.mapperNodeMap.get(name);
87    }
88
89    @Override
90    public SqlSession openSession() {
91      SqlSession newSqlSession = new DefaultSqlSession();
92      newSqlSession.setJdbcTemplate(jdbcTemplate);
93      newSqlSession.setSqlSessionFactory(this);
94
95      return newSqlSession;
96    }
97  }
```

使用 Sql Session 访问数据

有了上面的准备工作，上层的应用程序在使用的时候，就可以通过 Aurowired 注解直接拿到这个 SqlSessionFactory 了，然后通过工厂创建一个 Sql Session，再执行 SQL 命令。你可以看一下示例。

复制代码

```
package com.test.service;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import com.minis.batis.SqlSession;
import com.minis.batis.SqlSessionFactory;
import com.minis.beans.factory.annotation.Autowired;
import com.minis.jdbc.core.JdbcTemplate;
import com.minis.jdbc.core.RowMapper;
import com.test.entity.User;

public class UserService {
    @Autowired
    SqlSessionFactory sqlSessionFactory;

    public User getUserInfo(int userid) {
      //final String sql = "select id, name,birthday from users where id=?";
      String sqlid = "com.test.entity.User.getUserInfo";
      SqlSession sqlSession = sqlSessionFactory.openSession();
      return (User)sqlSession.selectOne(sqlid, new Object[]{new Integer(userid)},
          (pstmt)->{
            ResultSet rs = pstmt.executeQuery();
            User rtnUser = null;
            if (rs.next()) {
              rtnUser = new User();
              rtnUser.setId(userid);
              rtnUser.setName(rs.getString("name"));
              rtnUser.setBirthday(new java.util.Date(rs.getDate("birthday").getTi
            } else {
            }
            return rtnUser;
          }
      );
    }
  }
```

从代码里可以看出，程序基本上与以前直接用 JdbcTemplate 一样，只是变成通过 sqlSession.selectOne 来执行了。

这个 SqlSession 是由工厂生成的：`SqlSession sqlSession = sqlSessionFactory.openSession();`。你可以看一下它在 DefaultSqlSessionFactory 类中的定义。

复制代码

```
  public SqlSession openSession() {
    SqlSession newSqlSession = new DefaultSqlSession();
    newSqlSession.setJdbcTemplate(jdbcTemplate);
    newSqlSession.setSqlSessionFactory(this);

    return newSqlSession;
  }
```

由上面代码可见，这个 Sql Session 也就是对 JdbcTemplate 进行了一下包装。

定义接口：

复制代码

```
package com.minis.batis;

import com.minis.jdbc.core.JdbcTemplate;
import com.minis.jdbc.core.PreparedStatementCallback;

public interface SqlSession {
  void setJdbcTemplate(JdbcTemplate jdbcTemplate);
  void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory);
  Object selectOne(String sqlid, Object[] args, PreparedStatementCallback pstmtca
}
```

需要注意的是，我们是在 openSession() 的时候临时设置的 JdbcTemplate，而不是在 Factory 中设置的。这个设计留下了灵活性，意味着我们每一次真正执行某条 SQL 语句的时候可以替换这个 JdbcTemplate，这个时序的设计使动态数据源成为可能，这在读写分离的时候特别有用。

我们也默认给一个实现类 DefaultSqlSession。

复制代码

```
package com.minis.batis;

import javax.sql.DataSource;
import com.minis.jdbc.core.JdbcTemplate;
import com.minis.jdbc.core.PreparedStatementCallback;

public class DefaultSqlSession implements SqlSession{
  JdbcTemplate jdbcTemplate;
  public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
  }
  public JdbcTemplate getJdbcTemplate() {
    return this.jdbcTemplate;
  }
  SqlSessionFactory sqlSessionFactory;
  public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {
    this.sqlSessionFactory = sqlSessionFactory;
  }
  public SqlSessionFactory getSqlSessionFactory() {
    return this.sqlSessionFactory;
  }
  @Override
  public Object selectOne(String sqlid, Object[] args, PreparedStatementCallback
    String sql = this.sqlSessionFactory.getMapperNode(sqlid).getSql();
    return jdbcTemplate.query(sql, args, pstmtcallback);
  }

  private void buildParameter(){
  }

  private Object resultSet2Obj() {
    return null;
  }
}
```

这个 session 实现很单薄，对外就是一个 selectOne()，可以看出，程序最终还是落到了 jdbcTemplate.query(sql, args, pstmtcallback) 方法上，像一个洋葱一样一层层包起来的。但是原理的说明还是都反映出来了。

到这里，我们就实现了一个极简的 MyBatis。

小结

我们这节课仿照 MyBatis 将 SQL 语句进行了配置化。通过一个 SqlSessionFactory 解析配置文件，以一个 id 来代表使用的 SQL 语句。应用程序使用的时候，给 SqlSession 传入一个 SQL 的 id 号就可以执行。我们看到最后还是落到了 JdbcTemplate 方法中。

当然，这个是极简版本，远远没有实现 MyBatis 丰富的功能。比如现在只有 select 语句，没有 update；比如 SqlSession 对外只有一个 selectOne 接口，非常单薄；比如没有 SQL 数据集缓存，每次都要重新执行；比如没有读写分离的配置。当然，如何在这个极简版本的基础上进行扩展，就需要你动动脑筋，好好思考一下了。

还是那句老话，我们在一步步构建框架的过程中，主要学习的是搭建框架的思路，拆解部件，让专门的部件去处理专门的事情，让自己的框架具有扩展性。

完整源代码参见 https://github.com/YaleGuo/minis

课后题

学完这节课，我也给你留一道思考题。我们只是简单地实现了 select 语句的配置，如何扩展到 update 语句？还有进一步地，如何实现读写分离？比如说 select 的时候从一个数据库来取，update 的时候从另一个数据库来取。欢迎你在留言区与我交流讨论，也欢迎你把这节课分享给需要的朋友。我们下节课见！

精选留言 (1)

peter

2023-04-16 来自北京

请教老师两个问题啊：
Q1：Mapper配置的几个问题
在“Mapper配置”部分，有一个xml文件的说明：
<mapper namespace="com.test.entity.User">
<select id="getUserInfo" parameterType="java.lang.Integer" resultType="com.test.entity.User">

下面是文字说明：“这个配置中，也同样有基本的一些元素：SQL 类型、SQL 的 id”，
请问：namespace有什么用？文字说明中的“SQL 类型”对应哪个部分？是对应namespace吗？
namespace与resultType相同，是必须相同吗？还是可以不相同？

Q2：没有请求时候访问数据库。
SpringBoot项目，controller中自动注入service，service中自动注入Mapper。请求来了之后，由controller处理，controller调用自动注入的service，service再调用自动注入的Mapper，这是典型的ssm流程。
但是，现在有一个需求：软件启动后，需要访问数据库，此时并没有请求。我现在的实现方法是：controller的构造函数中使用JDBC访问数据库，是成功的。
问题：软件启动后，controller的构造函数执行了，说明controller被实例化了，此时service会自动注入吗？ （项目是两年前做的，当时一开始是尝试还用ssm来访问数据库，但好像失败了；印象中好像是service为null,就是说没有自动注入。）

作者回复: namespace区分不同的mapper，以免重名。sql类型是指select,update这些。
建议不要用controller访问数据库，层次不清。