链滴

MyBatis 3 初体验

作者：flowaters

原文链接：https://ld246.com/article/1531810317455

来源网站：链滴

简介

使用了MyBatis，在与数据库交互时，可以不必再为下面的事情烦恼：

1. JDBC代码: 设置Driver，得到Connection
2. 手动设置参数：设置JDBC参数
3. 解析结果集：解析ResultSet结果

MyBatis怎么解决的呢?

1. JDBC代码：使用xml配置
2. 手动设置参数：使用xml配置
3. 解析结果集：将数据库中的记录，映射到 Java接口 + POJO(Plain Old Java Object,普通)对象。

MyBatis是一个常用的ORM(Object Relational Mapping)框架。

第一个例子

准备数据源

MySQL 数据库

建表和准备测试数据

```
CREATE DATABASE test;

CREATE TABLE blog (
    id INT NOT NULL,
    author_id INT NOT NULL,
    title VARCHAR(255),
    PRIMARY KEY (id)
);


INSERT INTO blog (id,author_id,title) VALUES (1,101,'Jim Business');
INSERT INTO blog (id,author_id,title) VALUES (2,102,'Bally Slog');
```

Java的pom依赖

pom依赖：

```
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.4.6</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
```

```
        <version>8.0.11</version>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-core</artifactId>
        <version>1.2.3</version>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.2.3</version>
    </dependency>
```

本文写作时，最新的版本为3.4.6

项目结构

项目中共有6个文件，3个Java文件，3个资源文件。

```
.
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   │   └── org
│   │   │       └── apache
│   │   │           └── ibatis
│   │   │               ├── Blog.java
│   │   │               ├── BlogMapper.java
│   │   │               └── MainMybatis.java
│   │   └── resources
│   │       ├── BlogMapper.xml
│   │       ├── logback.xml
│   │       └── mybatis-config.xml
│   └── test
│       ├── java
│       └── resources
└── target
```

Blog

```
package org.apache.ibatis;

import java.io.Serializable;

public class Blog implements Serializable {

    private Integer id;

    private String title;

    public Integer getId() {
        return id;
```

```
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    @Override
    public String toString() {
        return "Blog [id=" + id + ", title=" + title + "]";
    }

}
```

BlogMapper

```
package org.apache.ibatis;

public interface BlogMapper {
    Blog selectBlog(Integer id);
}
```

MainMybatis

```
package org.apache.ibatis;

import java.io.InputStream;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class MainMybatis {

    public static void main(String[] args) throws Exception {
        String resource = "mybatis-config.xml";
        InputStream inputStream = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream)

        SqlSession session = sqlSessionFactory.openSession();
        try {
            BlogMapper mapper = session.getMapper(BlogMapper.class);
            Blog blog = mapper.selectBlog(1);
            System.out.println(blog);
        } finally {
```

```
            session.close();
        }
    }
}
```

BlogMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/
ybatis-3-mapper.dtd">
<mapper namespace="org.apache.ibatis.BlogMapper">
    <select id="selectBlog" resultType="org.apache.ibatis.Blog">
        select * from Blog where id = #{id}
    </select>
</mapper>
```

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>
                %d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n
            </Pattern>
        </layout>
    </appender>

    <logger name="org.apache.ibatis" level="debug" additivity="false">
        <appender-ref ref="STDOUT" />
    </logger>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/
td/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <property name="url" value="jdbc:mysql://localhost:3306/test" />
                <property name="username" value="note" />
                <property name="password" value="abeffect" />
            </dataSource>
```

```
        </environment>
    </environments>
    <mappers>
        <mapper resource="BlogMapper.xml" />
    </mappers>
</configuration>
```

执行效果

```
09:59:48,355 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resour
e [logback-test.xml]
09:59:48,355 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resour
e [logback.groovy]
09:59:48,355 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Found resource [logba
k.xml] at [file:/Users/note/eclipse-workspace/testMybatis/target/classes/logback.xml]
09:59:48,399 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - debug attribu
e not set
09:59:48,399 |-INFO in ch.qos.logback.core.joran.action.AppenderAction - About to instantiate
appender of type [ch.qos.logback.core.ConsoleAppender]
09:59:48,409 |-INFO in ch.qos.logback.core.joran.action.AppenderAction - Naming appender a
 [STDOUT]
09:59:48,461 |-WARN in ch.qos.logback.core.ConsoleAppender[STDOUT] - This appender no l
nger admits a layout as a sub-component, set an encoder instead.
09:59:48,461 |-WARN in ch.qos.logback.core.ConsoleAppender[STDOUT] - To ensure compati
ility, wrapping your layout in LayoutWrappingEncoder.
09:59:48,461 |-WARN in ch.qos.logback.core.ConsoleAppender[STDOUT] - See also http://log
ack.qos.ch/codes.html#layoutInsteadOfEncoder for details
09:59:48,462 |-INFO in ch.qos.logback.classic.joran.action.LoggerAction - Setting level of logg
r [org.apache.ibatis] to DEBUG
09:59:48,462 |-INFO in ch.qos.logback.classic.joran.action.LoggerAction - Setting additivity of
ogger [org.apache.ibatis] to false
09:59:48,462 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction - Attaching appe
der named [STDOUT] to Logger[org.apache.ibatis]
09:59:48,463 |-INFO in ch.qos.logback.classic.joran.action.RootLoggerAction - Setting level of
OOT logger to DEBUG
09:59:48,463 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction - Attaching appe
der named [STDOUT] to Logger[ROOT]
09:59:48,464 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - End of config
ration.
09:59:48,464 |-INFO in ch.qos.logback.classic.joran.JoranConfigurator@47d384ee - Registerin
 current configuration as safe fallback point

2018-06-10 09:59:48 [main] DEBUG org.apache.ibatis.logging.LogFactory -
            Logging initialized using 'class org.apache.ibatis.logging.slf4j.Slf4jImpl' adapter.
2018-06-10 09:59:48 [main] DEBUG o.a.i.d.pooled.PooledDataSource -
            PooledDataSource forcefully closed/removed all connections.
2018-06-10 09:59:48 [main] DEBUG o.a.i.d.pooled.PooledDataSource -
            PooledDataSource forcefully closed/removed all connections.
2018-06-10 09:59:48 [main] DEBUG o.a.i.d.pooled.PooledDataSource -
            PooledDataSource forcefully closed/removed all connections.
2018-06-10 09:59:48 [main] DEBUG o.a.i.d.pooled.PooledDataSource -
            PooledDataSource forcefully closed/removed all connections.
2018-06-10 09:59:48 [main] DEBUG o.a.i.t.jdbc.JdbcTransaction -
            Opening JDBC Connection
```

```
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj
jdbc.Driver'. The driver is automatically registered via the SPI and manual loading of the driver
class is generally unnecessary.
Sun Jun 10 09:59:48 CST 2018 WARN: Establishing SSL connection without server's identity ver
fication is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements
SL connection must be established by default if explicit option isn't set. For compliance with e
isting applications not using SSL the verifyServerCertificate property is set to 'false'. You need
either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststt
re for server certificate verification.
2018-06-10 09:59:49 [main] DEBUG o.a.i.d.pooled.PooledDataSource -
        Created connection 690339675.
2018-06-10 09:59:49 [main] DEBUG o.a.i.t.jdbc.JdbcTransaction -
        Setting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.ConnectionImp
@2925bf5b]
2018-06-10 09:59:49 [main] DEBUG o.a.ibatis.BlogMapper.selectBlog -
        ==>  Preparing: select * from Blog where id = ?
2018-06-10 09:59:49 [main] DEBUG o.a.ibatis.BlogMapper.selectBlog -
        ==> Parameters: 1(Integer)
2018-06-10 09:59:49 [main] DEBUG o.a.ibatis.BlogMapper.selectBlog -
        <==      Total: 1
Blog [id=1, title=Jim Business]
```

MyBatis执行过程

在完成了第一个例子之后，回过头来，继续看其是怎么执行的。

构建SqlSessionFactory

MyBatis都是从SqlSessionFactory开始的。

SqlSessionFactory实例通过SqlSessionFactoryBuilder来构建得到。

可以通过XML来构建，如上面的例子。

也可以通过Configuration实例来构建，这个细节这里略过。这兴趣直接阅读不使用 XML 构建 SqlSessionFactory。

获取SqlSession，执行SQL方法

SqlSessionFactory，就是制造SqlSession的工厂类。

而SqlSession，包含了操作数据库的所有SQL方法。

这些方法都是在配置文件中已经映射过的SQL方法。

如示例中映射过的selectBlog方法。

具体的代码片断为：

```
BlogMapper mapper = session.getMapper(BlogMapper.class);
Blog blog = mapper.selectBlog(1);
```

其它特征

通过注解映射SQL语句

第一个示例中，将BlogMapper中的接口方法，映射到了BlogMapper.xml中的mapper实现中。

这样写略微有些繁琐。

MyBatis也支持在接口类中，直接通过注解的方式来写实现的。

举个例子吧。这里增加一个BlogMapper2接口类，实现与BlogMapper同样的效果。

增加类文件BlogMapper2.class

```
package org.apache.ibatis;

import org.apache.ibatis.annotations.Select;

public interface BlogMapper2 {
    @Select("SELECT * FROM blog WHERE id = #{id}")
    Blog selectBlog(Integer id);
}
```

其中包含有一个@Select注解，实现了BlogMapper.xml的效果。

此外修改一下SqlSessionFactory的配置文件mybatis-config.xml，注册对应的Mapper。否则会报常“Type interface is not known to the MapperRegistry”。

修改后的片断如下：

```
    <mappers>
        <mapper resource="BlogMapper.xml" />
        <mapper class="org.apache.ibatis.BlogMapper2" />
    </mappers>
```

MainMybatis中也修改一下：

```
        SqlSession session = sqlSessionFactory.openSession();
        try {
            BlogMapper2 mapper = session.getMapper(BlogMapper2.class);
            Blog blog = mapper.selectBlog(1);
            System.out.println(blog);
        } finally {
            session.close();
        }
```

执行效果片断如下：

```
2018-06-10 11:07:30 [main] DEBUG o.a.i.d.pooled.PooledDataSource -
            Created connection 2079179914.
2018-06-10 11:07:30 [main] DEBUG o.a.i.t.jdbc.JdbcTransaction -
            Setting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.ConnectionImp
@7bedc48a]
2018-06-10 11:07:30 [main] DEBUG o.a.ibatis.BlogMapper2.selectBlog -
            ==>  Preparing: SELECT * FROM blog WHERE id = ?
2018-06-10 11:07:30 [main] DEBUG o.a.ibatis.BlogMapper2.selectBlog -
            ==> Parameters: 1(Integer)
```

```
2018-06-10 11:07:30 [main] DEBUG o.a.ibatis.BlogMapper2.selectBlog -
                <==      Total: 1
Blog [id=1, title=Jim Business]
```

更详细的使用介绍

作用域(Scope)

SqlSessionFactoryBuilder

在构建完SqlSessionFactory后就没用了。

SqlSessionFactory

单例

SqlSession

SqlSession不是线程安全的，不能被共享。在请求后，要及时的关闭。

如

```
SqlSession session = sqlSessionFactory.openSession();
try {
  // do work
} finally {
  session.close();
}
```

Mapper

mapper是从SqlSession中得到的，其作用域和SqlSession是相同的。

如

```
SqlSession session = sqlSessionFactory.openSession();
try {
  BlogMapper mapper = session.getMapper(BlogMapper.class);
  // do work
} finally {
  session.close();
}
```

XML配置文件

详细见官方文档 XML 映射配置文件

这里记录几个细节

加载外部配置文件

```
<properties resource="org/mybatis/example/config.properties">
  <property name="username" value="dev_user"/>
  <property name="password" value="F2Fa3!33TYyg"/>
</properties>

<dataSource type="POOLED">
  <property name="driver" value="${driver}"/>
  <property name="url" value="${url}"/>
  <property name="username" value="${username}"/>
  <property name="password" value="${password}"/>
</dataSource>
```

这样就直接把配置文件写在外面的config.properties中了，而不用每次修改xml文件。

另一种加载外部配置文件方法

也可以在构造SqlSessionFactory时，将properties做为参数传入。如：

```
SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(reader, props);
```

或者

```
SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(reader, environment, props);
```

这几种方式中，优先级从高到低为：

- 从build参数中传入
- 从resource属性读入
- properties元素体内指定

其它

还可以：

- 设置属性的默认值
- 使用三元运算符

具体看官方文档

settings

这里记录一部分，完整请看官方文档。

设置参数	描述	默认值
mapUnderscoreToCamelCase	是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 A_COLUMN 到经典 Java 属性名 aColumn 的类似映射	false
defaultExecutorType	配置默认的执行器。SIMPLE 就是普通的执器；REUSE 执行器会重用预处理语句（prepared statements）； BATCH 执行器将重用语句并执行量更新。	SIMPLE

示例如下：

```
<settings>
  <setting name="mapUnderscoreToCamelCase" value="false"/>
  <setting name="defaultExecutorType" value="SIMPLE"/>
</settings>
```

映射器(mappers)

使用相对于类路径的资源引用

```
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>
```

使用完全限定资源定位符URL

```
<mappers>
  <mapper url="file:///var/mappers/AuthorMapper.xml"/>
  <mapper url="file:///var/mappers/BlogMapper.xml"/>
  <mapper url="file:///var/mappers/PostMapper.xml"/>
</mappers>
```

使用映射器接口实现类的完全限定类名

```
<mappers>
  <mapper class="org.mybatis.builder.AuthorMapper"/>
  <mapper class="org.mybatis.builder.BlogMapper"/>
  <mapper class="org.mybatis.builder.PostMapper"/>
</mappers>
```

将包内的映射器接口实现全部注册为映射器

```
<mappers>
  <package name="org.mybatis.builder"/>
</mappers>
```

其它

- typeAliases: 为 Java 类型设置一个短的名字
- typeHandlers: 转换JDBC类型和Java类型的值
- 处理枚举类型: 使用EnumTypeHandler或者EnumOrdinalTypeHandler
- 对象工厂(objectFactory): 创建结果对象的新实例的工厂
- 插件(plugins): 可以在已映射语句执行过程中的某一点进行拦截调用
- 配置环境(environments): 每个数据库对应一个 SqlSessionFactory 实例
- databaseIdProvider: 根据不同的数据库厂商执行不同的语句

参考

- MyBatis 3 | 入门
- 针对PostgreSQL的最佳Java ORM框架
- mybatis hello world

附录

本文写作背景

提供MySQL兼容接口

Palo是百度开源的一款OLAP引擎，提供MySQL兼容的接口。

这样用户在使用的时候，不需要关心其内容实现过程，将其当成MySQL来使用即可。

阿里云的HybirdDB for MySQL也是一款类似的产品，同样提供了MySQL兼容的接口。

最近在调研如何把我们的服务也以类似的方式以MySQL兼容接口提供给用户。

调用MySQL兼容接口

在最初的最初，先以调用方的视角，看看调用方怎么调用MySQL的吧。

因此，本文从最广泛使用的MyBatis切入，来体验了一下如何调用MySQL服务。