第6章 MyBatis的核心配置

主讲人：陈向

湖南科技大学
计算机科学与工程学院

学习目标/Target

- 了解MyBatis核心对象的作用
- 掌握MyBatis核心配置文件及其元素的使用
- 掌握MyBatis映射文件及其元素的使用

章节概述/ Summary

通过上一章的学习，读者对MyBatis框架已经有了一个初步了解，但是要想熟练地使用MyBatis框架进行实际开发，只会简单的配置是不行的，我们还需要对框架中的核心对象、核心配置文件以及映射文件有更深入的了解。本章将针对MyBatis核心对象、核心配置文件和映射文件进行讲解。

目录/Contents

01 MyBatis的核心对象

02 MyBatis核心配置文件

03 MyBatis映射文件

04 案例：员工管理系统

6.1 MyBatis的核心对象

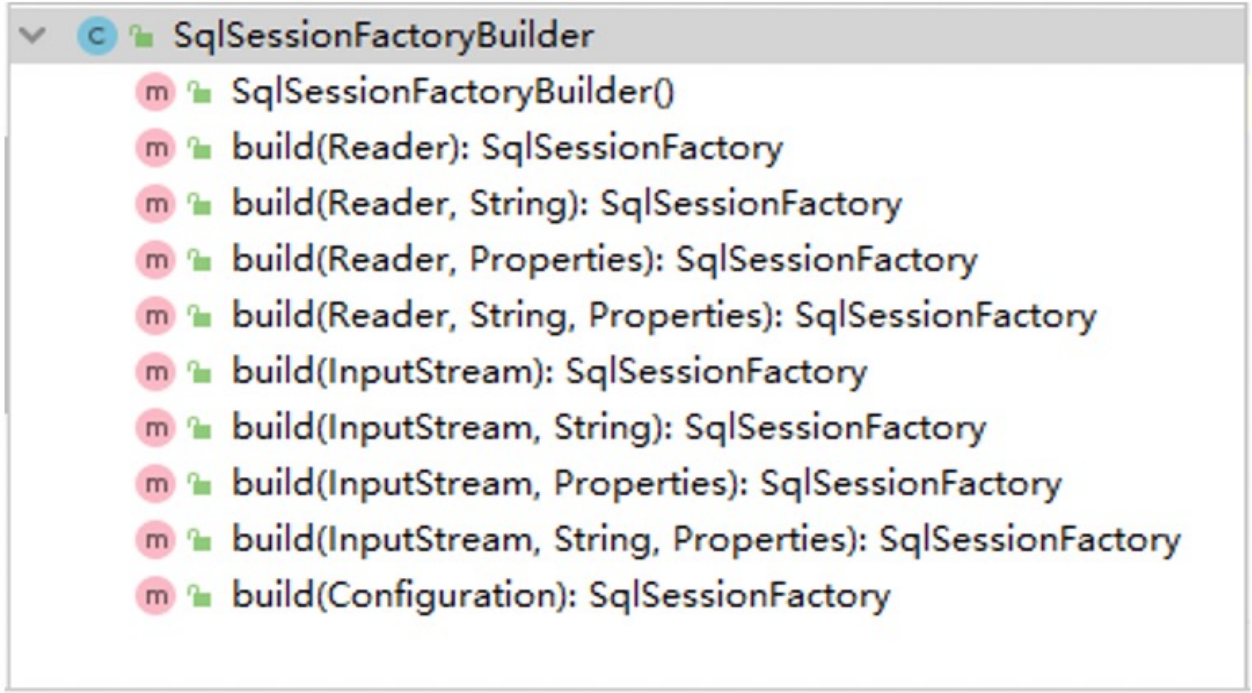
6.1.1 SqlSessionFactoryBuilder

- 了解Mybatis框架的核心对象SqlSessionFactoryBuilder，能够说出它的作用和特点

6.1.1 SqlSessionFactoryBuilder

SqlSessionFactoryBuilder的多个重载build()方法

建造者模式



6.1.1 SqlSessionFactoryBuilder

SqlSessionFactoryBuilder构建build()方法的形式

由于build()方法中的参数environment和properties都可以为null，所以SqlSessionFactoryBuilder构建SqlSessionFactory对象的build()方法按照配置信息的传入方式，可以分为三种形式。

6.1.1 SqlSessionFactoryBuilder

形式一：SqlSessionFactoryBuilder构建build()方法

```
build(InputStream inputStream,String environment,Properties properties)
```

上述build()方法中，参数inputStream是字节流，它封装了XML文件形式的配置信息；参数environment和参数properties为可选参数。其中，参数environment决定将要加载的环境，包括数据源和事务管理器；参数properties决定将要加载的properties文件。

6.1.1 SqlSessionFactoryBuilder

形式二：SqlSessionFactoryBuilder构建build()方法

```
build(Reader reader,String environment,Properties properties)
```

由上述build()方法可知，第二种形式的build()方法参数作用与第一种形式大体一致，唯一不同的是，第一种形式的build()方法使用InputStream字节流封装了XML文件形式的配置信息，而第二种形式的build()方法使用Reader字符流封装了xml文件形式的配置信息。

6.1.1 SqlSessionFactoryBuilder

形式三：SqlSessionFactoryBuilder构建build()方法

```
build(Configuration config)
```

通过以上代码可知，配置信息可以通过InputStream（字节流）、Reader（字符流）、Configuration（类）三种形式提供给SqlSessionFactoryBuilder的build()方法。

6.1.1 SqlSessionFactoryBuilder

以读取XML文件的方式构造SqlSessionFactory对象

通过过读取XML配置文件的方式构造SqlSessionFactory对象的关键代码如下所示。

```
// 读取配置文件
InputStream inputStream = Resources.getResourceAsStream("配置文件位置");
// 根据配置文件构建SqlSessionFactory
SqlSessionFactory sqlSessionFactory =
new SqlSessionFactoryBuilder().build(inputStream);
```

6.1.1 SqlSessionFactoryBuilder

使用什么模式创建SqlSessionFactory对象

SqlSessionFactory对象是线程安全的，它一旦被创建，在整个应用程序执行期间都会存在。如果我们多次创建同一个数据库的SqlSessionFactory对象，那么该数据库的资源将很容易被耗尽。通常每一个数据库都只创建一个SqlSessionFactory对象，所以在构建SqlSessionFactory对象时，建议使用单例模式。

6.1.2 SqlSessionFactory

- 了解Mybatis框架的核心对象SqlSessionFactory，能够说出它的作用和特点

6.1.2 SqlSessionFactory

SqlSessionFactory的openSession()方法

方法名称	描述
SqlSession openSession()	开启一个事务。
SqlSession openSession(Boolean autoCommit)	参数autoCommit可设置是否开启事务。
SqlSession openSession(Connection connection)	参数connection可提供自定义连接。
SqlSession openSession(TransactionIsolationLevel level)	参数level可设置隔离级别。
SqlSession openSession(ExecutorType execType)	参数execType有三个可选值。
SqlSession openSession(ExecutorType execType , Boolean autoCommit)	参数execType有三个可选值。
SqlSession openSession(ExecutorType execType , Connection connection)	参数ExecutorType有三个可选值。

6.1.2 SqlSessionFactory

openSession(ExecutorType execType)参数值

参数execType有三个可选值:

- ExecutorType.SIMPLE：表示为每条语句创建一条新的预处理语句。
- ExecutorType.REUSE：表示会复用预处理语句。
- ExecutorType.BATCH：表示会批量执行所有更新语句。

6.1.2 SqlSessionFactory

openSession(ExecutorType execType，Boolean autoCommit)参数值

- 参数execType有三个可选值，同openSession(ExecutorType execType)的参数。
- 参数autoCommit可设置是否开启事务。

6.1.2 SqlSessionFactory

openSession(ExecutorType execType，Connection connection)参数值

- 参数execType有三个可选值，同openSession(ExecutorType execType)的参数。
- 参数connection可提供自定义连接。

6.1.3 SqlSession

- 掌握Mybatis框架的核心对象SqlSession，能够使用SqlSession的常用方法执行SQL操作

6.1.3 SqlSession

SqlSession对象的作用

SqlSession是MyBatis框架中另一个重要的对象，它是应用程序与持久层之间执行交互操作的一个单线程对象，主要作用是执行持久化操作，类似于JDBC中的Connection。SqlSession对象包含了执行SQL操作的方法，由于其底层封装了JDBC连接，所以可以直接使用SqlSession对象来执行已映射的SQL语句。

6.1.3 SqlSession

SqlSession对象中常用方法

方法名称	描述
<T> T selectOne(String statement)	查询方法。参数statement是在配置文件中定义的<select>元素的id。
<T> T selectOne(String statement, Object parameter)	查询方法。parameter是查询语句所需的参数。
<E> List<E> selectList(String statement)	查询方法。参数statement是在配置文件中定义的<select>元素的id。
<E> List<E> selectList(String statement, Object parameter)	查询方法。parameter是查询语句所需的参数。
<E> List<E> selectList(String statement, Object parameter, RowBounds rowBounds)	查询方法。rowBounds是用于分页的参数对象。
void select(String statement, Object parameter, ResultHandler handler)	查询方法。 ，handler对象用于处理查询语句返回的复杂结果集。

6.1.3 SqlSession

SqlSession对象中常用方法

方法名称	描述
int insert(String statement)	插入方法。参数statement是在配置文件中定义的<insert>元素的id。
int insert(String statement, Object parameter)	插入方法。parameter是插入语句所需的参数。
int update(String statement)	更新方法。参数statement是在配置文件中定义的<update>元素的id。
int update(String statement, Object parameter)	更新方法。parameter是更新语句所需的参数。
int delete(String statement)	删除方法。参数statement是在配置文件中定义的<delete>元素的id。
int delete(String statement, Object parameter)	删除方法。parameter是删除语句所需的参数。

6.1.3 SqlSession

SqlSession对象中常用方法

方法名称	描述
void commit()	提交事务的方法。
void rollback()	回滚事务的方法。
void close()	关闭SqlSession对象。
<T> T getMapper(Class<T> type)	该方法会返回Mapper接口的代理对象。参数type是Mapper的接口类型。
Connection getConnection()	获取JDBC数据库连接对象的方法。

6.1.3 SqlSession

SqlSession对象的使用范围

每一个线程都应该有一个自己的SqlSession对象，并且该对象不能共享。SqlSession对象是线程不安全的，因此其使用范围最好在一次请求或一个方法中，绝不能将其放在类的静态字段、对象字段或任何类型的管理范围（如Servlet的HttpSession）中使用。SqlSession对象使用完之后，要及时的关闭，SqlSession对象通常放在finally块中关闭，代码如下所示。

```
SqlSession sqlSession = sqlSessionFactory.openSession();
try {
        // 此处执行持久化操作
} finally {        sqlSession.close();        }
```

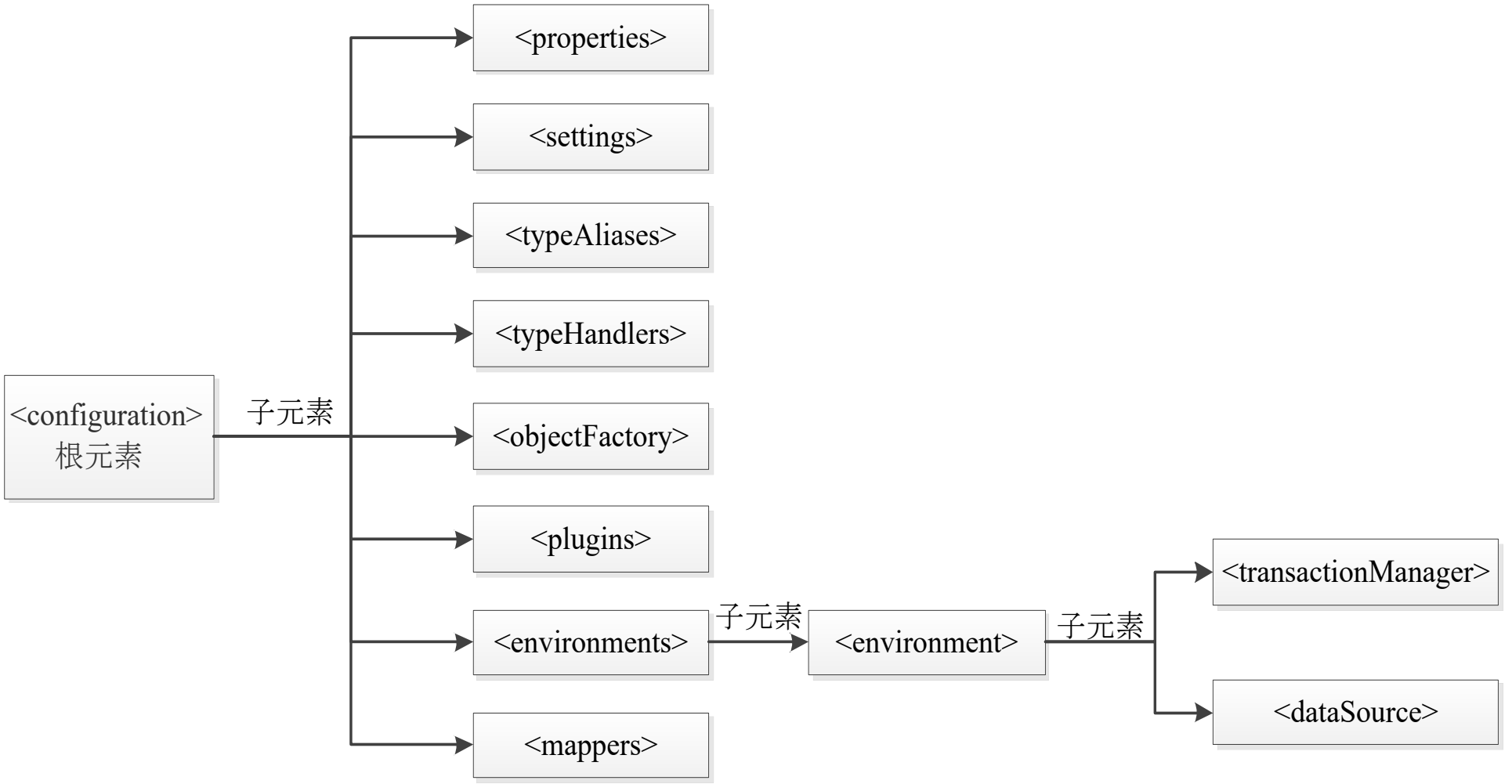
6.2 MyBatis核心配置文件

6.2.1 配置文件的主要元素

- 熟悉Mybatis配置文件的主要元素，能够说出主要元素都有哪些

6.2.1 配置文件的主要元素

MyBatis核心配置文件中的主要元素



6.2.1 配置文件的主要元素

<configuration>的子元素的执行顺序

<configuration>元素是整个XML配置文件的根元素，相当于MyBatis各元素的管理员。<configuration>有很多子元素，MyBatis的核心配置就是通过这些子元素完成的。需要注意的是，在核心配置文件中，<configuration>的子元素必须按照上图由上到下的顺序进行配置，否则MyBatis在解析XML配置文件的时候会报错。

6.2.2 <properties>元素

- 掌握Mybatis配置文件的<properties>元素，能够使用<properties>元素读取外部文件的配置信息

6.2.2 <properties>元素

<properties>是一个配置属性的元素，该元素的作用是读取外部文件的配置信息。

假设现在有一个配置文件 db.properties，该文件配置了数据库的连接信息，具体如下：

```
jdbc.driver=com.mysql.cj.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/mybatis
jdbc.username=root
jdbc.password=root
```

6.2.2 <properties>元素

如果想获取数据库的连接信息，可以在 MyBatis 的核心配置文件 mybatis-config.xml 中使用<properties>元素先引入 db.properties 文件，具体代码如下：

```
<properties resource="db.properties" />
```

6.2.2 <properties>元素

引入 db.properties 文件后，如果希望动态获取 db.properties 文件中的数据库连接信息，可以使用<property>元素配置，示例代码如下：

```
<dataSource type="POOLED">
    <!-- 数据库驱动 -->
    <property name="driver" value="${jdbc.driver}" />
    <!-- 连接数据库的url -->
    <property name="url" value="${jdbc.url}" />
    <!-- 连接数据库的用户名 -->
    <property name="username" value="${jdbc.username}" />
    <!-- 连接数据库的密码 -->
    <property name="password" value="${jdbc.password}" />
</dataSource>
```

6.2.2 <properties>元素

db.properties文件实现动态参数配置

完成上述配置后，<dataSource>元素中连接数据库的 4 个属性（driver、url、username 和 password）值将会由db.properties 文件中对应的值来动态替换。这样一来，<properties>元素就可以通过 db.properties 文件实现动态参数配置。

6.2.3 <settings>元素

- 掌握Mybatis配置文件的<settings>元素，能够使用<settings>元素开启缓存和开启延迟加载

6.2.3 <settings>元素

<settings>元素中的常见配置参数

配置参数	描述
cacheEnabled	用于配置是否开启缓存。
lazyLoadingEnabled	延迟加载的全局开关。
aggressiveLazyLoading	关联对象属性的延迟加载开关。
multipleResultSetsEnabled	是否允许单一语句返回多结果集（需要兼容驱动）。
useColumnLabel	使用列标签代替列名。
useGeneratedKeys	允许JDBC支持自动生成主键，需要驱动兼容。

6.2.3 <settings>元素

<settings>元素中的常见配置参数

配置参数	描述
autoMappingBehavior	指定MyBatis应如何自动映射列到字段或属性。
defaultExecutorType	配置默认的执行器。
defaultStatementTimeout	配置超时时间，它决定驱动等待数据库响应的秒数。
mapUnderscoreToCamelCase	是否开启自动驼峰命名规则（camel case）映射。
jdbcTypeForNull	当没有为参数提供特定的JDBC类型时，为空值指定JDBC类型。

6.2.3 <settings>元素

<settings>元素中常见配置参数的使用方式

```
<settings>
    <!-- 是否开启缓存 -->
    <setting name="cacheEnabled" value="true" />
    <!-- 是否开启延迟加载,如果开启,所有关联对象都会延迟加载 -->
    <setting name="lazyLoadingEnabled" value="true" />
    <!-- 是否开启关联对象属性的延迟加载,如果开启,对任意延迟属性的调用都
    会使用带有延迟加载属性的对象向完整加载,否则每种属性都按需加载 -->
    <setting name="aggressiveLazyLoading" value="true" />
    ...
</settings>
```

6.2.4 <typeAliases>元素

- 掌握Mybatis配置文件的<typeAliases>元素，能够使用<typeAliases>元素设置别名

6.2.4 <typeAliases>元素

核心配置文件若要引用一个POJO实体类，需要输入POJO实体类的全限定类名，而全限定类名比较冗长，如果直接输入，很容易拼写错误。例如，POJO实体类User的全限定类名是com.itheima.pojo.User，未设置别名之前，映射文件的select语句块若要引用POJO类User，必须使用其全限定类名，引用代码如下。

```
<select id="findById" parameterType="int" resultType="com.itheima.pojo.User">
    select * from users where uid = #{id}
</select>
```

6.2.4 <typeAliases>元素

多个全限定类设置别名的方式

方式一：在<typeAliases>元素下，使用多个<typeAlias>元素为每一个全限定类逐个配置别名。

```
<typeAliases>
    <typeAlias alias= “Usertype="com.itheima.pojo.User"/>
    <typeAlias alias="Student" type="com.itheima.pojo.Student"/>
    <typeAlias alias="Employee"
               type="com.itheima.pojo.Employee"/>
    <typeAlias alias="Animal" type="com.itheima.pojo.Animal"/>
</typeAliases>
```

6.2.4 <typeAliases>元素

多个全限定类设置别名的方式

方式二：通过自动扫描包的形式自定义别名。

```
<typeAliases>
    <package name="com.itheima.pojo"/>
</typeAliases>
```

6.2.4 <typeAliases>元素

常见Java类型的默认别名问题

除了可以使用<typeAliases>元素为实体类自定义别名外，MyBatis框架还为许多常见的Java类型（如数值、字符串、日期和集合等）提供了相应的默认别名。例如别名_byte映射类型byte、_long映射类型long等，别名可以在MyBatis中直接使用，但由于别名不区分大小写，所以在使用时要注意重复定义的覆盖问题。

6.2.5 <environments>元素

- 掌握Mybatis配置文件的<environments>元素，能够使用<environments>元素定义运行环境

6.2.5 <environments>元素

<environments>元素配置运行环境

MyBatis可以配置多套运行环境，如开发环境、测试环境、生产环境等，我们可以灵活选择不同的配置，从而将SQL映射到不同运行环境的数据库中。不同的运行环境可以通过<environments>元素来配置，但不管增加几套运行环境，都必须要明确选择出当前要用的唯一的一个运行环境。

6.2.5 <environments>元素

<environments>各元素配置运行环境

MyBatis的运行环境信息包括事务管理器和数据源。在MyBatis的核心配置文件中，MyBatis通过<environments>元素定义一个运行环境。<environment>元素有两个子元素，<transactionManager>元素和<daraSource>元素。<transactionManager>元素用于配置运行环境的事务管理器；<daraSource>元素用于配置运行环境的数据源信息。

6.2.5 <environments>元素

使用<environments>元素进行配置的示例代码

```
<environments default="development">
    <environment id="development">
        <transactionManager type="JDBC" /><!—设置使用JDBC事务管理 -->
        <dataSource type="POOLED"> <!-配置数据源 -->
            <property name="driver" value="${jdbc.driver}" />
            <property name="url" value="${jdbc.url}" />
            <property name="username" value="${jdbc.username}" />
            <property name="password" value="${jdbc.password}" />
        </dataSource>
    </environment>          ...
</environments>
```

6.2.5 <environments>元素

<transcationManager>元素配置事务管理器

在MyBatis中，<transcationManager>元素可以配置两种类型的事务管理器，分别是JDBC和MANAGED。

- JDBC：此配置直接使用JDBC的提交和回滚设置，它依赖于从数据源得到的连接来管理事务的作用域。
- MANAGED：此配置不提交或回滚一个连接，而是让容器来管理事务的整个生命周期。默认情况下，它会关闭连接，但可以将<transcationManager>元素的closeConnection属性设置为false来阻止它默认的关闭行为。

6.2.5 <environments>元素

MyBatis数据源类型

项目中使用Spring+MyBatis，则没必要在MyBatis中配置事务管理器，实际开发中，项目会使用Spring自带的管理器来实现事务管理。对于数据源的配置，MyBatis提供了UNPOOLED、POOLED和JNDI三种数据源类型。

6.2.5 <environments>元素

UNPOOLED数据源

UNPOOLED表示数据源为无连接池类型。配置此数据源类型后，程序在每次被请求时会打开和关闭数据库连接。UNPOOLED适用于对性能要求不高的简单应用程。UNPOOLED类型的数据源需要配置5种属性。

6.2.5 <environments>元素

UNPOOLED数据源需要配置的属性

属性	说明
driver	JDBC驱动的Java类的完全限定名
url	数据库的URL地址
username	登录数据库的用户名
password	登录数据库的密码
defaultTransactionIsolationLevel	默认的连接事务隔离级别

6.2.5 <environments>元素

POOLED数据源

POOLED表示数据源为连接池类型。POOLED数据源利用“池”的概念将JDBC连接对象组织起来，节省了在创建新的连接对象时需要初始化和认证的时间。POOLED数据源使得并发Web应用可以快速的响应请求，是当前比较流行的数据源配置类型。

6.2.5 <environments>元素

POOLED数据源可额外配置的属性

属性	说明
poolMaximumActiveConnections	在任意时间可以存在的活动连接数量，默认值：10。
poolMaximumIdleConnections	任意时间可能存在的空闲连接数。
poolMaximumCheckoutTime	在被强制返回之前，池中连接被检出时间，默认值：20000毫秒。
poolTimeToWait	如果获取连接花费的时间较长，它会给连接池打印状态日志并重新尝试获取一个连接，默认值：20000毫秒。
poolPingQuery	发送到数据库的侦测查询，用来检验连接是否处在正常工作秩序中。默认是“NO PING QUERY SET”。
poolPingEnabled	是否启用侦测查询，默认值：false。
poolPingConnectionsNotUsedFor	配置poolPingQuery的使用频度。

6.2.5 <environments>元素

JNDI数据源

JNDI 表示数据源可以在 EJB 或应用服务器等容器中使用。JNDI数据源需要配置的属性如下所示。

属性	说明
initial_context	该属性主要用于在InitialContext中寻找上下文。该属性为可选属性，在忽略时，data_source属性会直接从InitialContext中寻找。
data_source	该属性表示引用数据源对象位置的上下文路径。如果提供了initial_context配置，那么程序会在其返回的上下文中进行查找；如果没有提供，则直接在InitialContext中查找。

6.2.6 <mappers>元素

- 掌握Mybatis配置文件的<mappers>元素，能够使用<mappers>元素引入 MyBatis 映射文件

6.2.6 <mappers>元素

<mappers>元素作用

在MyBatis的核心配置文件中，<mappers>元素用于引入MyBatis映射文件。映射文件包含了POJO对象和数据表之间的映射信息，MyBatis通过核心配置文件中的<mappers>元素找到映射文件并解析其中的映射信息。通过<mappers>元素引入映射文件的方法有4种。

6.2.6 <mappers>元素

a.使用类路径引入

使用类路径引入映射文件的示例代码如下所示。

```
<mappers>
    <mapper resource="com/itheima/mapper/UserMapper.xml"/>
</mappers>
```

6.2.6 <mappers>元素

b.使用本地文件路径引入

使用本地文件路径引入映射文件的示例代码如下所示。

```
<mappers>
    <mapper url="file:///D:/com/itheima/mapper/UserMapper.xml"/>
</mappers>
```

6.2.6 <mappers>元素

c.使用接口类引入

使用接口类引入映射文件的示例代码如下所示。

```
<mappers>
    <mapper class="com.itheima.mapper.UserMapper"/>
</mappers>
```

6.2.6 <mappers>元素

d.使用包名引入

使用包名引入映射文件的示例代码如下所示。

```
<mappers>
    <package name="com.itheima.mapper"/>
</mappers>
```

6.3 MyBatis映射文件

6.3.1 MyBatis映射文件中的常用元素

- 掌握Mybatis映射文件中的常用元素，能够使用MyBatis 映射文件中的常用元素编写语句

6.3.1 MyBatis映射文件中的常用元素

MyBatis映射文件中的常用元素

属性	说明
mapper	映射文件的根元素，该元素只有一个namespace属性（命名空间）。
cache	配置给定命名空间的缓存。
cache-ref	从其他命名空间引用缓存配置。
resultMap	描述数据库结果集和对象的对应关系。
sql	可以重用的SQL块，也可以被其他语句使用。
insert	用于映射插入语句。
delete	用于映射删除语句。
update	用于映射更新语句。
select	用于映射查询语句。

6.3.1 MyBatis映射文件中的常用元素

<mapper>元素中namespace属性作用

namespace属性有两个作用：

- 用于区分不同的mapper，全局唯一。
- 绑定DAO接口，即面向接口编程。当namespace绑定某一接口之后，可以不用写该接口的实现类，MyBatis会通过接口的全限定名查找到对应的mapper配置来执行SQL语句，因此namespace的命名必须跟接口同名。

6.3.1 MyBatis映射文件中的常用元素

<mapper>元素如何区别不同的XML文件

在不同的映射文件中，<mapper>元素的子元素的id可以相同，MyBatis通过<mapper>元素的namespace属性值和子元素的id联合区分不同的Mapper.xml文件。接口中的方法与映射文件中SQL语句id应一一对应。

6.3.2 <select>元素

- 掌握Mybatis映射文件的<select>元素，能够使用<select>元素执行查询操作

6.3.2 <select>元素

<select>元素的查询使用

<select>元素用来映射查询语句，它可以从数据库中查询数据并返回。使用<select>元素执行查询操作非常简单，示例代码如下：

```
<!—查询操作 -->
<select id="findUserById" parameterType="Integer"
    resultType="com.itheima.pojo.User">
    select * from users where id = #{id}
</select>
```

6.3.2 <select>元素

<select>元素的常用属性

属性	说明
id	表示命名空间中<select>元素的唯一标识，通过该标识可以调用这条查询语句。
parameterType	它是一个可选属性，用于指定SQL语句所需参数类的全限定名或者别名，其默认值是unset。
resultType	用于指定执行这条SQL语句返回的全限定类名或别名。
resultMap	表示外部resultMap的命名引用。resultMap和resultType不能同时使用。
flushCache	用于指定是否需要MyBatis清空本地缓存和二级缓存。

6.3.2 <select>元素

<select>元素的常用属性

属性	说明
useCache	用于控制二级缓存的开启和关闭。
timeout	用于设置超时时间，单位为秒。
fetchSize	获取记录的总条数设定，默认值是unset。
statementType	用于设置MyBatis预处理类。
resultSetType	表示结果集的类型，它的默认值是unset。

6.3.3 <insert>元素

- 掌握Mybatis映射文件的<insert>元素，能够使用<insert>元素执行插入操作

6.3.3 <insert>元素

<insert>元素的插入使用

<insert>元素用于映射插入语句，在执行完<insert>元素中定义的SQL语句后，会返回插入记录的数量。使用< insert >元素执行插入操作非常简单，示例代码如下：

```
<!—插入操作 --><insert id="addUser"
parameterType="com.itheima.pojo.User">
    insert into
users(uid,uname,uage)values(#{uid},#{uname},#{uage})
</insert>
```

6.3.3 <insert>元素

数据库获取主键值的方式

很多时候，执行插入操作后，需要获取插入成功的数据生成的主键值，不同类型数据库获取主键值的方式不同，下面分别对支持主键自动增长的数据库获取主键值和不支持主键自动增长的数据库获取主键值的方式进行介绍。

2.3.3 <insert>元素

a.使用支持主键自动增长的数据库获取主键值

如果使用的数据库支持主键自动增长（如MySQL和SQL Server），那么可以通过keyProperty属性指定POJO类的某个属性接收主键返回值（通常会设置到id属性上），然后将useGeneratedKeys的属性值设置为true。

```
<insert id="addUser" parameterType="com.itheima.pojo.User"
        keyProperty="uid" useGeneratedKeys="true" >
        insert into
        users(uid,uname,uage)values(#{uid},#{uname},#{uage})
</insert>
```

6.3.3 <insert>元素

b.使用不支持主键自动增长的数据库获取主键值

使用MyBatis提供的<selectKey>元素来自定义主键。

```
<selectKey
    keyProperty="id"          resultType="Integer"
    order="BEFORE"            statementType="PREPARED">
```

在上述<selectKey>元素的属性中，order属性可以被设置为BEFORE或AFTER。如果设置为BEFORE，那么它会首先执行<selectKey>元素中的配置来设置主键，然后执行插入语句；如果设置为AFTER，那么它先执行插入语句，然后执行<selectKey>元素中的配置内容。

6.3.4 <update>元素

- 掌握Mybatis映射文件的<update>元素，能够使用<update>元素执行更新操作

6.3.4 <update>元素

<update>元素的更新使用

<update>元素用于映射更新语句，它可以更新数据库中的数据。在执行完元素中定义的SQL语句后，会返回更新的记录数量。使用<update>元素执行更新操作非常简单，示例代码如下：

```
<!—更新操作 -->
<update id="updateUser"
parameterType="com.itheima.pojo.User">
    update users set uname= #{uname},uage = #{uage} where
uid = #{uid}          </update>
```

6.3.5 <delete>元素

- 掌握Mybatis映射文件的<delete>元素，能够使用<delete>元素执行删除操作

6.3.5 <delete>元素

<delete>元素的删除使用

<delete>元素用于映射删除语句，在执行完<delete>元素中的SQL语句之后，会返回删除的记录数量。使用<delete>元素执行删除操作非常简单，示例代码如下所示：

```
<!-- 删除操作 -->
<delete id="deleteUser" parameterType="Integer">
    delete from users where uid=#{uid}
</delete>
```

<delete>元素中，除了上述示例代码中的几个属性外，还有其他一些可以配置的属性，如flushCache、timeout等。

6.3.5 <delete>元素

<delete>元素中的属性

属性	说明
id	表示命名空间中<select>元素的唯一标识，通过该标识可以调用这条语句。
parameterType	它是一个可选属性，用于指定SQL语句所需参数类的全限定名或者别名，其默认值是unset。
flushCache	用于指定是否需要MyBatis清空本地缓存和二级缓存。
timeout	用于设置超时时间，单位为秒。
statementType	用于设置MyBatis预处理类。

6.3.6 <sql>元素

- 掌握Mybatis映射文件的<sql>元素，能够使用<sql>元素定义可重用的 SQL 代码片段

6.3.6 <sql>元素

<sql>元素的作用

在一个映射文件中，通常需要定义多条SQL语句，这些SQL语句的组成可能有一部分是相同的（如多条select语句中都查询相同的id、username字段），如果每一个SQL语句都重写一遍相同的部分，势必会增加代码量。针对此问题，可以在映射文件中使用MyBatis所提供的<sql>元素，将这些SQL语句中相同的组成部分抽取出来，然后在需要的地方引用。

<sql>元素的作用是定义可重用的SQL代码片段，它可以被包含在其他语句中。<sql>元素可以被静态地（在加载参数时）参数化，<sql>元素不同的属性值通过包含的对象发生变化。

6.3.6 <sql>元素

实现一个根据客户id查询客户信息的SQL片段

```
<!--定义要查询的表 -->
<sql id= “someinclude">from <include refid="${include_target}" /></sql>
<!--定义查询列 --><sql id= “userColumns">    uid,uname,uage    </sql>
<!--根据客户id查询客户信息 -->
<select id="findUserById" parameterType="Integer"
        resultType="com.itheima.pojo.User">      select
    <include refid="userColumns"/>
    <include refid="someinclude">
        <property name="include_target" value="users" /></include>
    where uid = #{uid}
</select>
```

6.3.7 <resultMap>元素

- 掌握Mybatis映射文件中的<resultMap>元素，能够使用<resultMap>元素定义映射规则

6.3.7 <resultMap>元素

<resultMap>元素的作用

<resultMap>元素表示结果映射集，是MyBatis中最重要也是功能最强大的元素。<resultMap>元素主要作用是定义映射规则、更新级联以及定义类型转化器等。

数据表中的列和需要返回的对象的属性可能不会完全一致，这种情况下MyBatis不会自动赋值，这时就需要使用<resultMap>元素进行结果集映射。

6.3.7 <resultMap>元素

接下来通过一个具体的案例演示使用<resultMap>元素进行结果集映射，具体步骤如下。

STEP 01 在名称为mybatis的数据库中，创建一个t_student表，并插入几条测试数据。

```
USE mybatis;
CREATE TABLE t_student(
   sid INT PRIMARY KEY AUTO_INCREMENT,
   sname VARCHAR(50),
   sage INT
);
INSERT INTO t_student(sname,sage) VALUES('Lucy',25);
INSERT INTO t_student(sname,sage) VALUES('Lili',20);
INSERT INTO t_student(sname,sage) VALUES('Jim',20);
```

6.3.7 <resultMap>元素

STEP 02

创建实体类Student，用于封装学生信息。在类中定义id、name和age属性，以及属性的getter/setter方法和toString()方法。

```
package com.itheima.pojo;
public class Student {
    private Integer id;          // 主键id
    private String name;         // 学生姓名
    private Integer age;         // 学生年龄
    // 省略getter/setter方法
     @Override
     public String toString() {
return "User [id=" + id + ", name=" + name + ", age=" + age + "]";}
}
```

6.3.7 <resultMap>元素

STEP 03

创建映射文件StudentMapper.xml，在映射文件中编写映射查询语句。

```
<！-- 只显示mapper元素的内容-->
<mapper namespace="com.itheima.mapper.StudentMapper">
    <resultMap type="com.itheima.pojo.Student" id="studentMap">
    <id property="id" column="sid"/><result property="name"
            column="sname"/><result property="age"
column="sage"/>
    </resultMap>
    <select id="findAllStudent" resultMap="studentMap">
            select * from t_student</select>
</mapper>
```

6.3.7 <resultMap>元素

STEP 04

在核心配置文件mybatis-config.xml中，引入StudentMapper.xml，将StudentMapper.xml映射文件加载到程序中。在mybatis-config.xml中的<mapper>元素下添加如下代码。

```
<mapper
resource="com/itheima/mapper/StudentMapper.xml">
<mapper>
```

6.3.7 <resultMap>元素

STEP 05

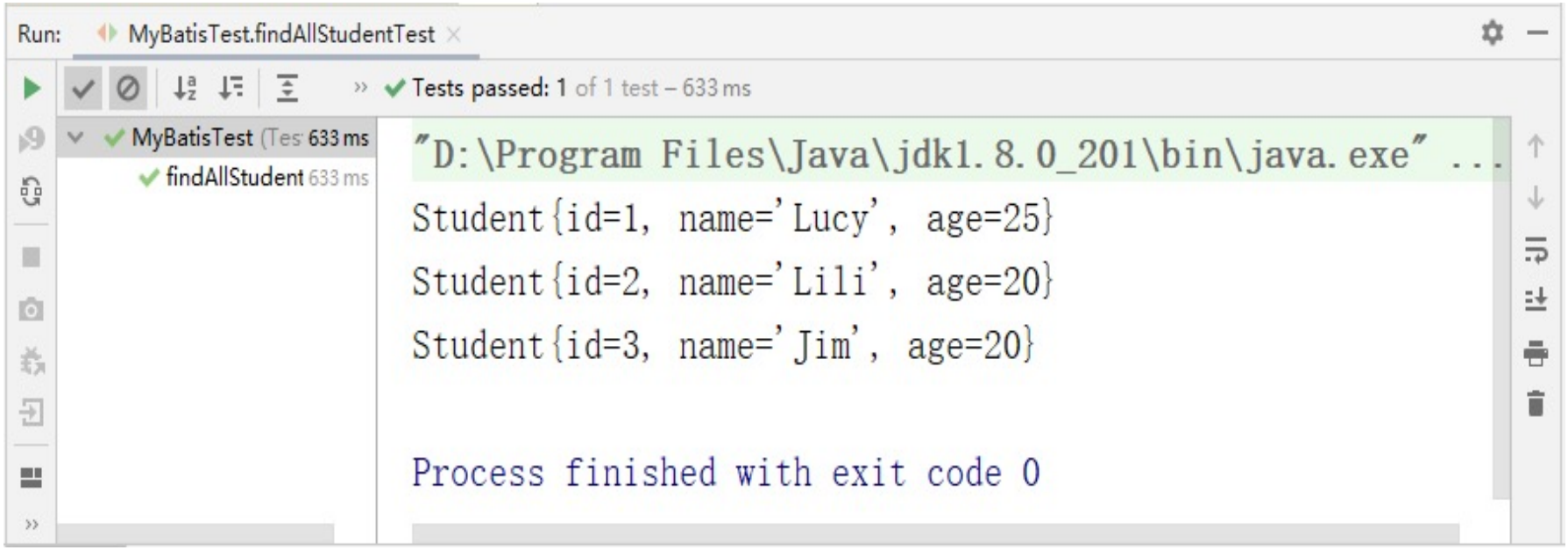
创建测试类MyBatisTest，在测试类中，编写测试方法findAllStudentTest()，用于测试<resultMap>元素实现查询结果的映射。

```
public class MyBatisTest {
    private SqlSessionFactory sqlSessionFactory;
    private SqlSession sqlSession;
    // init()方法省略
    @Test
    public void findAllStudentTest() {
        List<Student> list =
                sqlSession.selectList("com.itheima.mapper.StudentMapper.
                findAllStudent");
        for (Student student : list) {        System.out.println(student);}}
    // destory()方法省略
}
```

6.3.7 <resultMap>元素

STEP 06

运行MyBatisTest测试类，控制台会输出结果。



需要注意的是，在测试类MyBatisTest中，每一个用@Test注解标注的方法称为测试方法，他们的调用顺序为@Before→@Test→@After。

6.3.7 <resultMap>元素

多学一招：使用工具类创建SqlSession对象

在上述案例中，由于每个方法执行时都需要读取配置文件，并根据配置文件的信息构建SqlSessionFactory对象、创建SqlSession对象、释放资源，这导致了大量的重复代码。为了简化开发，我们可以将读取配置文件和释放资源的代码封装到一个工具类中，然后通过工具类创建SqlSession对象。

6.4 案例：员工管理系统

6.4 案例：员工管理系统

- 完成一个员工管理系统，能够实现如下功能根据id查询、新增、修改、删除员工信息

6.4 案例：员工管理系统

员工表详情

现有一张员工表如下。利用本章所学知识完成一个员工管理系统。实现如下功能：根据id查询员工信息、新增员工信息、根据id修改员工信息、根据id删除员工信息。

员工编号（id）	商品名称（name）	员工年龄（age）	员工职位（position）
1	张三	20	员工
2	李四	18	员工
3	王五	35	经理

6.4 案例：员工管理系统

案例要求

本案例要求根据员工表在数据库中创建一个employee表，并利用本章所学知识完成一个员工管理系统，该系统需要实现以下几个功能：

根据id查询员工信息；新增员工信息；

根据id修改员工信息；根据id删除员工信息。

6.4 案例：员工管理系统

STEP 01

项目搭建：创建建一个名称为mybatisdemo的项目，并在项目中引入 MySQL 驱动包、JUnit 测试包、MyBatis 的核心包等相关依赖、创建数据库连接信息配置文件、创建 MyBatis 的核心配置文件。核心配置文件的内容如下。

```
<configuration><properties resource="db.properties"/>
<environments default="development">
<environment id="development">
        <transactionManager type="JDBC"/>
         <dataSource type="POOLED">
              <property name="driver" value="${mysql.driver}" />
              <property name="url" value="${mysql.url}" />
              <property name="username" value="${mysql.username}" />
              <property name="password" value="${mysql.password}" />
         </dataSource>
    </environment></environments>
</configuration>
```

6.4 案例：员工管理系统

STEP 02

数据准备：在mybatis数据库中创建employee表，并在employee表中插入几条数据。

```
use mybatis;
create table employee(
    id int primary key auto_increment,
    name varchar(20) not null,
    age int not null,
    position varchar(20) );
insert into employee(id,name,age,position) values(null,'张三',20,'员工  '),
        (null,'李四',18, '员工'),(null,'王五',35,'经理');
```

6.4 案例：员工管理系统

STEP 03

POJO类准备：创建持久化类Employee，并在类中声明id（编号）、name（姓名）、age（年龄）和position（职位）属性，以及属性对应的getter/setter方法。

```
public class Employee {
    private Integer id;          private String name;
    private Integer age;
    private String position;
    // 省略getter/setter方法
    @Override
    public String toString() {
        return "Employee{" + "id=" + id + ", name=" + name +
                ", age=" + age + ", position=" + position +'}’ ;
}}
```

6.4 案例：员工管理系统

STEP 04

编写映射文件：创建映射文件EmployeeMapper.xml，该文件主要用于实现SQL语句和Java对象之间的映射，部分文件内容如下。

```
<mapper namespace="com.itheima.mapper.EmployeeMapper">
<select id="findById"parameterType="Integer”
    resultType="com.itheima.pojo.Employee"> select * from
employee where id = #{id}
</select>
<insert id="addEmployee"
parameterType="com.itheima.pojo.Employee">
    insert into employee(id,name,age,position)values
            (#{id},#{name},#{age},#{position})
</insert> </mapper>
```

STEP 05

修改mybatis-config.xml核心配置文件：在mybatis-config.xml映射文件的<mappers>元素下添加EmployeeMapper.xml映射文件路径的配置，用于将EmployeeMapper.xml映射文件加载到程序中。

```
<mapper
resource="com/itheima/mapper/EmployeeMapper.xml">
</mapper>
```

6.4 案例：员工管理系统

STEP 06

编写MyBatisUtils工具类 ：创建MyBatisUtils工具类，该类用于封装读取配置文件信息的代码。

```
public class MyBatisUtils {
    private static SqlSessionFactory sqlSessionFactory = null;
    static {            try {
            // 使用MyBatis提供的Resources类加载MyBatis的配置文件
            Reader reader = Resources.getResourceAsReader("mybatis-
config.xml");
            // 构建SqlSessionFactory工厂
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
    } catch (Exception e) { e.printStackTrace();}}
    public static SqlSession getSession() {//获取SqlSession对象的静态方法
            return sqlSessionFactory.openSession();}
}
```

STEP 07

编写测试类

（1）在项目src/test/java目录下创建Test包，在Test包下创建MyBatisTest测试类，用于程序测试。在MyBatisTest测试类中添加findByIdTest()方法，用于根据id查询员工信息。

（2）在MyBatisTest测试类中添加insertEmployeeTest()方法，用于插入员工信息。

（3）在MyBatisTest测试类中添加updateEmployeeTest()方法，用于更新员工信息。

（4）在MyBatisTest测试类中添加deleteEmployeeTest()方法，用于删除员工信息。

本章小结

本章主要对MyBatis的核心配置进行了详细讲解。首先讲解了MyBatis中的三个重要核心对象SqlSessionFactoryBuilder、SqlSessionFactory和SqlSession；然后介绍了核心配置文件中的元素及其使用；最后对映射文件中的几个主要元素进行了详细讲解。通过本章的学习，读者将能够了解MyBatis中三个核心对象的作用，熟悉核心配置文件中常用元素的使用，并掌握映射文件中常用元素的使用。