第 1 章 MyBatis 快速入门

1.1 ORM 简介

面向对象程序设计是企业级开发常用的设计方式，我们在实践中常用的编程语言，如 Java、.Net、C++等，都是面向对象的编程语言。在实际生产环境中常用的数据库产品，如 MySQL、Oracle 等，则都是关系型数据库。虽然 NoSQL 数据库，如 HBase、MongoDB、Couchbase 等，在最近一段时间有了飞速的发展，也有一部分互联网应用开始尝试使用 NoSQL 数据库管理其部分数据，但是关系型数据库凭借多年的发展和技术积累，以及众多成功的案例等优势，依然占据着市场的主要地位。

在系统开发过程中，开发人员需要使用面向对象的思维实现业务逻辑，但设计数据库表或是操作数据库记录时，则需要通过关系型的思维方式考虑问题。应用程序与关系型数据库之间进行交互时，数据在对象和关系结构中的表、列、字段等之间进行转换。

JDBC 是 Java 与数据库交互的统一 API，实际上它分为两组 API，一组是面向 Java 应用程序开发人员的 API，另一组是面向数据库驱动程序开发人员的 API。前者是一个标准的 Java API 且独立于各个厂家的数据库实现，后者则是数据库驱动程序开发人员用于编写数据库驱动，是前者的底层支持，一般与具体的数据库产品相关。

在实际开发 Java 系统时，我们可以通过 JDBC 完成多种数据库操作。这里以传统 JDBC 编程中的查询操作为例进行说明，其主要步骤如下：

（1）注册数据库驱动类，明确指定数据库 URL 地址、数据库用户名、密码等连接信息。

（2）通过 DriverManager 打开数据库连接。

（3）通过数据库连接创建 Statement 对象。

（4）通过 Statement 对象执行 SQL 语句，得到 ResultSet 对象。

（5）通过 ResultSet 读取数据，并将数据转换成 JavaBean 对象。

（6）关闭 ResultSet、Statement 对象以及数据库连接，释放相关资源。

上述步骤 1～步骤 4 以及步骤 6 在每次查询操作中都会出现，在保存、更新、删除等其他数据库操作中也有类似的重复性代码。在实践中，为了提高代码的可维护性，可以将上述重复性代码封装到一个类似 DBUtils 的工具类中。步骤 5 中完成了关系模型到对象模型的转换，要使用比较通用的方式封装这种复杂的转换是比较困难的。

为了解决该问题，ORM（Object Relational Mapping，对象-关系映射）框架应运而生。如图 1-1 所示，ORM 框架的主要功能就是根据映射配置文件，完成数据在对象模型与关系模型之间的映射，同时也屏蔽了上述重复的代码，只暴露简单的 API 供开发人员使用。

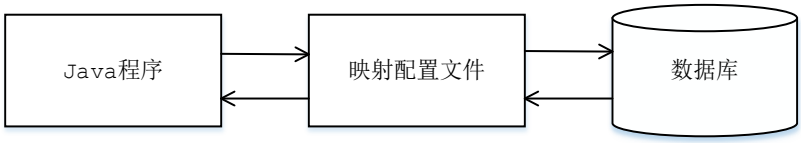

图 1-1

另外，实际生产环境中对系统的性能是有一定要求的，数据库作为系统中比较珍贵的资源，极易成为整个系统的性能瓶颈，所以我们不能像上述 JDBC 操作那样简单粗暴地直接访问数据库、直接关闭数据库连接。应用程序一般需要通过集成缓存、数据源、数据库连接池等组件进行优化，如果没有 ORM 框架的存在，就要求开发人员熟悉相关组件的 API 并手动编写集成相关的代码，这就提高了开发难度并延长了开发周期。

很多 ORM 框架都提供了集成第三方缓存、第三方数据源等组件的接口，而且这些接口都是业界统一的，开发和运维人员可以通过简单的配置完成第三方组件的集成。当系统需要更换第三方组件时，只要选择支持该接口的组件并更新配置即可，这不仅提高了开发效率，而且提高了系统的可维护性。

最后，建议读者在开发大中型项目时，优先考虑使用 ORM 框架，并根据下文的介绍选择合适的 ORM 框架。之所以这么说，是因为笔者在设计某些项目时，未使用 ORM 框架，到项目的中后期时为了便于项目的扩展和维护，使用各种设计模式等相关知识对程序的 DAO 层（Data Access Object，数据访问对象）进行了重构，最后得到一个不完善的、类似于 ORM 框架的设计。这也算是笔者实践中的一个教训，希望对读者有一定参考价值。

1.2 常见持久化框架

Hibernate

Hibernate 是一款 Java 世界中最著名的 ORM 框架之一，笔者撰稿时 Hibernate 的最新版本是 5.2 版本。作为一个老牌的 ORM 框架，Hibernate 经受住了 Java EE 企业级应用的考验，替代了复杂的 Java EE 中 EJB 解决方案，一度成为 Java ORM 领域的首选框架。

Hibernate 通过 hbm.xml 映射文件维护 Java 类与数据库表的映射关系。通过 Hibernate 的映射，Java 开发人员可以用看待 Java 对象的角度去看待数据库表中的数据行。数据库中所有的表通过 hbm.xml 配置文件映射之后，都对应一个 Java 类，表中的每一行数据在运行过程中会被映射成相应的 Java 对象。在 Java 对象之间存在一对多、一对一、多对多等复杂的层次关系，Hibernate 的 hbm.xml 映射文件也可以维护这种层次关系，并将这种关系与数据库中的外键、关联表等进行映射，这也就是所谓的“关联映射”。

例如，一个用户（User）可以创建多个订单（Order），而一个订单（Order）只属于一个用户，两者之间存在一对多的关系。在 Java 代码中可以在 User 类中添加一个 List<Order>类型的字段来维护这种一对多关系，在数据库中可以在订单表（t_order）中添加一个 user_id 列作为外键，指向用户表（t_user）的主键 id，从而维护这种一对多的关系，如图 1-2 所示。

在 Hibernate 中，可以通过如下 User.hbm.xml 配置文件将这两种关系进行映射。

```
<hibernate-mapping>
    <!-- 表和类之间的映射 -->
    <class name="com.xxx.User" table="t_user">
        <!-- 主键映射 -->
        <id name="id" column="id"/>
        <!-- 属性映射 -->
        <property name="name" column="name"/>
        <!-- 表之间关系映射 -->
        <set name="orders" cascade="save-update,delete">
            <key column="user_id"/>
            <one-to-many class="com.xxx.Order"/>
        </set>
    </class>
</hibernate-mapping>
```

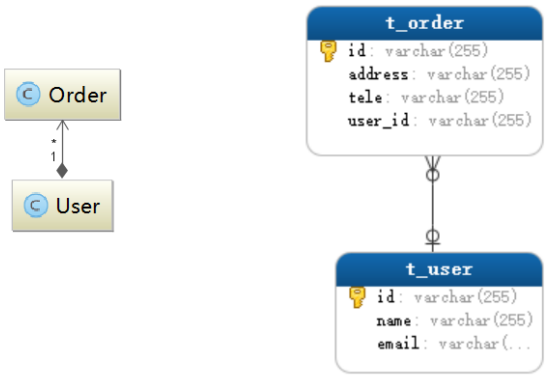

图 1-2

如果是双向关联，则在 Order 中添加 User 类型的字段指向关联的 User 对象，并使用相应的 Order.hbm.xml 配置文件进行配置，示例如下：

```
<hibernate-mapping>
    <!-- 表和类之间的映射 -->
    <class name="com.xxx.Order" table="t_order">
        <!-- 主键映射 -->
        <id name="id" column="id"/>
        <!-- 属性映射 -->
        <property name="address" column="address"/>
        <property name="tele" column="tele"/>
        <!-- 表之间关系映射 -->
        <many-to-one name="user" column="user_id"></many-to-one>
    </class>
</hibernate-mapping>
```

一对一、多对多等关联映射与上述配置类似，这里就不再赘述，感兴趣的读者可以参考 Hibernate 的相关资料进行学习。

Hibernate 除了能够实现对象模型与关系模型的映射，还可以帮助开发人员屏蔽不同数据库产品中 SQL 语句的细微差异。现在不同的关系型数据库产品对 SQL 标准的支持不尽相同，这就会出现同一条 SQL 语句在 MySQL 上可以正常执行，而在 Oracle 数据库上执行报错的情况。Hibernate 封装了数据库层面的全部操作，开发人员不再需要直接编写 SQL 语句，只需要使用 Hibernate 提供的简单易懂的 API 即可完成数据库操作。例如，Hibernate 提供的 Criteria 是一个完全面向对象、可扩展的条件查询 API，使用它查询数据库时完全不需要考虑数据库底层如何

实现、SQL 语句如何编写。除了 Criteria，Hibernate 还提供了一种称为 HQL（Hibernate Query Language）的语言，从语句的结构上来看，HQL 语句与 SQL 语句十分类似，但它是一种面向对象的查询语言。对于复杂的数据库查询，开发人员可以按照面向对象的思维方式编写 HQL 实现，Hibernate 会根据实际配置的数据库方言，将 HQL 语句生成对应的 SQL 语句。Hibernate 通过其简洁的 API 以及统一的 HQL 语句，帮助上层程序屏蔽掉底层数据库的差异，增强了程序的可移植性。

另外，Hibernate 的 API 没有侵入性，业务逻辑不需要继承 Hibernate 的任何接口。Hibernate 默认提供了一级缓存和二级缓存，这有利于提高系统的性能，降低数据库压力。Hibernate 还有其他的特性和优点，例如，支持透明的持久化、延迟加载、由对象模型自动生成数据库表等，感兴趣的读者可以查阅 Hibernate 的相关资料进行学习。

但是，Hibernate 并不是一颗万能药。数据库本身有自己的组织方式，并不是数据库中所有的概念都能在面向对象的世界中找到合适的映射，例如，索引、存储过程、函数等，尤其是索引，它对数据库查询的性能帮助很大，适当优化 SQL 语句，选择使用合适的索引会提高整个查询的速度。但是，我们很难修改 Hibernate 生成的 SQL 语句，当数据量比较大、数据库结构比较复杂时，Hibernate 生成 SQL 语句会非常复杂，而且要让生成的 SQL 语句使用正确的索引也比较困难，这就会导致出现大量慢查询的情况。在有些大数据量、高并发、低延迟的场景下，Hibernate 并不是特别适合。最后，Hibernate 对批处理的支持并不是很友好，这也会影响部分性能。后来出现了 iBatis（Mybatis 的前身）这种半自动化的映射方式来解决性能问题。

JPA

JPA（Java Persistence API）是 EJB 3.0 中持久化部分的规范，但它可以脱离 EJB 的体系单独作为一个持久化规范进行使用。Gavin King 作为 Hibernate 创始人，同时也参与了 JPA 规范的编写，所以在 JPA 规范中可以看到很多与 Hibernate 类似的概念和设计。这里需要读者了解的是，JPA 仅仅是一个持久化的规范，它并没有提供具体的实现。其他持久化厂商会提供 JPA 规范的具体实现，例如，Hibernate、EclipseLink 等都提供了 JPA 规范的具体实现。JPA 规范的愿景很美好，但是并没有得到很好的发展，现在在实践中的出场率也不是很高。如果读者对 JPA 感兴趣，可以查阅相关资料进行学习，这里就不再做过多介绍了。

Spring JDBC

严格来说，Spring JDBC 并不能算是一个 ORM 框架，它仅仅是使用模板方式对原生 JDBC 进行了一层非常薄的封装。使用 Spring JDBC 可以帮助开发人员屏蔽创建数据库连接对、Statement 对象、异常处理以及事务管理的重复性代码，提高开发效率。

Spring JDBC 中没有映射文件、对象查询语言、缓存等概念，而是直接执行原生 SQL 语句。Spring JDBC 中提供了多种 Template 类，可以将对象中的属性映射成 SQL 语句中绑定的参数，

Spring JDBC 还提供了很多 ORM 化的 Callback，这些 Callback 可以将 ResultSet 转化成相应的对象列表。在有些场景中，我们需要直接使用 JDBC 原生对象，例如，操作 JDBC 原生的 ResultSet，则可以直接返回 SqlRowSet 对象，该对象是原生 ResultSet 对象的简单封装。Spring JDBC 在功能上不及 Hibernate 强大，但它凭借高度的灵活性，也在 Java 持久化中占有了一席之地。

除此之外，Spring JDBC 本身就位于 Spring 核心包中，也是 Spring 框架的基础模块之一，天生与 Spring 框架无缝集成。凭借 Spring 框架的强大功能，Spring JDBC 可以实现集成多种开源数据源、管理声明式事务等功能。总的来说，Spring JDBC 可以算作一个封装良好、功能强大的 JDBC 工具集。Spring JDBC 整体架构设计非常优秀，其源码也非常值得分析，感兴趣的读者可以深入学习一下。

MyBatis

最后要压轴登场的是本书的主角——MyBatis。MyBatis 前身是 Apache 基金会的开源项目 iBatis，在 2010 年该项目脱离 Apache 基金会并正式更名为 MyBatis。在 2013 年 11 月，MyBatis 迁移到了 GitHub。

MyBatis 与前面介绍的持久化框架一样，可以帮助开发人员屏蔽底层重复性的原生 JDBC 代码。MyBatis 通过映射配置文件或相应注解将 ResultSet 映射为 Java 对象，其映射规则可以嵌套其他映射规则以及子查询，从而实现复杂的映射逻辑，也可以实现一对一、一对多、多对多映射以及双向映射。

相较于 Hibernate，MyBatis 更加轻量级，可控性也更高，在使用 MyBatis 时我们直接在映射配置文件中编写待执行的原生 SQL 语句，这就给了我们直接优化 SQL 语句的机会，让 SQL 语句选择合适的索引，能更好地提高系统的性能，比较适合大数据量、高并发等场景。在编写 SQL 语句时，我们也可以比较方便地指定查询返回的列，而不是查询所有列并映射对象后返回，这在列比较多的时候也能起到一定的优化效果。

在实际业务中，对同一数据集的查询条件可能是动态变化的，如果读者有使用 JDBC 或其他类似框架的经历就能体会到，根据不同条件拼接 SQL 语句是一件非常麻烦的事情，尤其是拼接过程中要确保在合适的位置添加“where”、“and”、“in”等 SQL 语句的关键字以及空格、逗号、等号等分隔符，而且这个拼接过程非常枯燥、没有技术含量，可能经过反复调试才能得到一个可执行的 SQL 语句。MyBatis 提供了强大的动态 SQL 功能来帮助开发人员摆脱这种窘境，开发人员只需要在映射配置文件中编写好动态 SQL 语句，MyBatis 就可以根据执行时传入的实际参数值拼凑出完整的、可执行的 SQL 语句。

通过上面的介绍，我们对常见的持久化框架有了一定认识，那我们如何选择合适的持久化框架呢？从性能角度来看，Hibernate 生成的 SQL 语句难以优化，Spring JDBC 和 MyBatis 直接使用原生 SQL 语句，优化空间比较大，MyBatis 和 Hibernate 有设计良好的缓存机制，三者都可

以与第三方数据源配合使用；从可移植性角度来看，Hibernate 帮助开发人员屏蔽了底层数据库方言，而 Spring JDBC 和 MyBatis 在该方面没有做很好的支持，但实践中很少有项目会来回切换底层使用的数据库产品，所以这点并不是特别重要；从开发效率的角度来看，Hibernate 和 MyBatis 都提供了 XML 映射配置文件和注解两种方式实现映射，Spring JDBC 则是通过 ORM 化的 Callback 的方式进行映射。读者在进行技术选型时，可以从更多角度进行比较，权衡性能、可扩展性、开发人员技术栈等多个方面选择合适的框架。

1.3 MyBatis 示例

在开始介绍 MyBatis 整体架构之前，先来通过一个 MyBatis 示例帮助读者快速了解 MyBatis 中常见的概念。首先来看 mybatis-config.xml 配置文件，这是 MyBatis 中的基础配置文件，其中配置了数据库的 URL 地址、数据库用户名和密码、别名信息、映射配置文件的位置以及一些全局配置信息，如下所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration  PUBLIC ... >
<configuration>
    <properties> <!-- 定义属性值 -->
        <property name="username" value="root"/>
        <property name="id" value="123"/>
    </properties>
    <settings><!-- 全局配置信息 -->
        <setting name="cacheEnabled" value="true"/>
        ... ...
    </settings>
    <typeAliases>
        <!-- 配置别名信息，在映射配置文件中可以直接使用 Blog 这个别名代替 com.xxx.Blog 这个类 -->
        <typeAlias type="com.xxx.Blog" alias="Blog"/>
        ... ...
    </typeAliases>
    <environments default="development">
        <environment id="development">
            <!-- 配置事务管理器的类型 -->
            <transactionManager type="JDBC"/>
            <!-- 配置数据源的类型，以及数据库连接的相关信息 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
```

```
                <property name="url" value="jdbc:mysql://localhost:3306/test"/>
                <property name="username" value="root"/>
                <property name="password" value=""/>
            </dataSource>
        </environment>
    </environments>
    <!-- 配置映射配置文件的位置 -->
    <mappers>
        <mapper resource="com/xxx/BlogMapper.xml"/>
    </mappers>
</configuration>
```

了解了 mybatis-config.xml 配置文件的大致结构之后，我们来看一下 BlogMapper.xml 映射配置文件的结构，具体代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" ... >
<mapper namespace="com.xxx.BlogMapper">
    <!-- 定义映射规则 -->
    <resultMap id="detailedBlogResultMap" type="Blog">
        <constructor> <!-- 构造函数映射 -->
            <idArg column="blog_id" javaType="int"/>
        </constructor>
        <!--  属性映射 -->
        <result property="title" column="blog_title"/>
        <!-- 对象属性的映射，同时也是一个嵌套映射，后面会详细分析嵌套映射的处理过程 -->
        <association property="author" resultMap="authorResult"/>
        <!-- 集合映射，也是一个匿名的嵌套映射 -->
        <collection property="posts" ofType="Post">
            <id property="id" column="post_id"/>
            <result property="content" column="post_content"/>
        </collection>
    </resultMap>

    <resultMap id="authorResult" type="Author">
        <id property="id" column="author_id"/>
        <result property="username" column="username"/>
        <result property="password" column="password"/>
```

```
        <result property="email" column="email"/>
    </resultMap>

    <!-- 定义 SQL 语句，除了 select 节点，还可以定义 insert、update、delete 节点。为了便于描述，
后面统称为“SQL 节点” -->
    <select id="selectBlogDetails" resultMap="detailedBlogResultMap">
        select B.id as blog_id, B.title as blog_title, B.author_id as blog_author_id,
        A.id as author_id, A.username as author_username, A.password as author_password,
        A.email as author_email, P.id as post_id, P.blog_id as post_blog_id,
        P.content as post_content
        from Blog B  left outer join Author A on B.author_id = A.id
        left outer join Post P on B.id = P.blog_id  where B.id = #{id}
    </select>
</mapper>
```

最后我们来看一下 Java 程序中如何加载上述配置文件以及如何使用 MyBatis 的 API。应用程序首先会加载 mybatis-config.xml 配置文件，并根据配置文件的内容创建 SqlSessionFactory 对象；然后，通过 SqlSessionFactory 对象创建 SqlSession 对象，SqlSession 接口中定义了执行 SQL 语句所需要的各种方法；之后，通过 SqlSession 对象执行映射配置文件中定义的 SQL 语句，完成相应的数据操作；最后，通过 SqlSession 对象提交事务，关闭 SqlSession 对象。整个过程的具体实现如下所示。

```
public class Main {
    public static void main(String[] args) throws Exception {
        String resource = "com/xxx/mybatis-config.xml";
        InputStream inputStream = Resources.getResourceAsStream(resource);
        // 加载 mybatis-config.xml 配置文件，并创建 SqlSessionFactory 对象
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder()
                .build(inputStream);
        // 创建 SqlSession 对象
        SqlSession session = sqlSessionFactory.openSession();
        try {
            Map<String, Object> parameter = new HashMap<>();
            parameter.put("id",1);
            // 执行 select 语句，将 ResultSet 映射成对象并返回
            Blog blog = (Blog) session.selectOne("com.xxx.BlogMapper.selectBlogDetails",
                    parameter);
            // 输出 Blog 对象
```

```
            System.out.println(blog);
        } finally {
            session.close();
        }
    }
}
```

上面涉及的 Blog、Author、Post 等都是普通的 JavaBean 对象，下面简略看一下这几个类的定义：

```
public class Blog implements Serializable {
    private int id;
    private String title;
    private Author author;
    private List<Post> posts;
    // ... 省略全部的 getter/setter 方法
}
public class Author implements Serializable{
    private int id;
    private String username;
    private String password;
    private String email;
    // ... 省略全部的 getter/setter 方法
}

public class Post {
    protected int id;
    protected Author author;
    protected String content;
    // ... 省略全部的 getter/setter 方法
}
```

本节介绍的示例是非常典型的 MyBatis 使用方式。为便于读者理解，在后面介绍 MyBatis 源代码时，还会以这种方式使用 MyBatis。

1.4 MyBatis 整体架构

MyBatis 的整体架构分为三层，分别是基础支持层、核心处理层和接口层，如图 1-3 所示。

图 1-3

1.4.1 基础支持层

基础支持层包含整个 MyBatis 的基础模块，这些模块为核心处理层的功能提供了良好的支撑。下面简单描述各个模块的功能，在第 2 章将会详细分析基础支持层中每个模块的实现原理。

- **反射模块**

  Java 中的反射虽然功能强大，但对大多数开发人员来说，写出高质量的反射代码还是有一定难度的。MyBatis 中专门提供了反射模块，该模块对 Java 原生的反射进行了良好的封装，提供了更加简洁易用的 API，方便上层使调用，并且对反射操作进行了一系列优化，例如缓存了类的元数据，提高了反射操作的性能。

- **类型转换模块**

  正如前面示例所示，MyBatis 为简化配置文件提供了别名机制，该机制是类型转换模块的主要功能之一。类型转换模块的另一个功能是实现 JDBC 类型与 Java 类型之间的转换，该功能在为 SQL 语句绑定实参以及映射查询结果集时都会涉及。在为 SQL 语句绑定实参时，会将数据由 Java 类型转换成 JDBC 类型；而在映射结果集时，会将数据由 JDBC 类型转换成 Java 类型。类型转换模块的具体原理在第 2 章详述。

- **日志模块**

  无论在开发测试环境中，还是在线上生产环境中，日志在整个系统中的地位都是非常重要的。良好的日志功能可以帮助开发人员和测试人员快速定位 Bug 代码，也可以帮助运维人员快速定位性能瓶颈等问题。目前的 Java 世界中存在很多优秀的日志框架，

例如 Log4j、Log4j2、slf4j 等。MyBatis 作为一个设计优良的框架，除了提供详细的日志输出信息，还要能够集成多种日志框架，其日志模块的一个主要功能就是集成第三方日志框架。

- **资源加载模块**

  资源加载模块主要是对类加载器进行封装，确定类加载器的使用顺序，并提供了加载类文件以及其他资源文件的功能。

- 解析器模块

  解析器模块的主要提供了两个功能：一个功能是对 XPath 进行封装，为 MyBatis 初始化时解析 mybatis-config.xml 配置文件以及映射配置文件提供支持；另一个功能是为处理动态 SQL 语句中的占位符提供支持。

- **数据源模块**

  数据源是实际开发中常用的组件之一。现在开源的数据源都提供了比较丰富的功能，例如，连接池功能、检测连接状态等，选择性能优秀的数据源组件对于提升 ORM 框架乃至整个应用的性能都是非常重要的。MyBatis 自身提供了相应的数据源实现，当然 MyBatis 也提供了与第三方数据源集成的接口，这些功能都位于数据源模块之中。在第 2 章会详细介绍该模块。

- **事务管理**

  MyBatis 对数据库中的事务进行了抽象，其自身提供了相应的事务接口和简单实现。在很多场景中，MyBatis 会与 Spring 框架集成，并由 Spring 框架管理事务，在第 4 章会介绍 MyBatis 如何与 Spring 集成开发，其中就会涉及 Spring 框架管理事务相关的配置。

- **缓存模块**

  在优化系统性能时，优化数据库性能是非常重要的一个环节，而添加缓存则是优化数据库时最有效的手段之一。正确、合理地使用缓存可以将一部分数据库请求拦截在缓存这一层，如图 1-4 所示，这就能够减少相当一部分数据库的压力。

  MyBatis 中提供了一级缓存和二级缓存，而这两级缓存都是依赖于基础支持层中的缓存模块实现的。这里需要读者注意的是，MyBatis 中自带的这两级缓存与 MyBatis 以及整个应用是运行在同一个 JVM 中的，共享同一块堆内存。如果这两级缓存中的数据量较大，则可能影响系统中其他功能的运行，所以当需要缓存大量数据时，优先考虑使用 Redis、Memcache 等缓存产品。

- Binding 模块

  通过前面的示例我们知道，在调用 SqlSession 相应方法执行数据库操作时，需要指定

映射文件中定义的 SQL 节点，如果出现拼写错误，我们只能在运行时才能发现相应的异常。为了尽早发现这种错误，MyBatis 通过 Binding 模块将用户自定义的 Mapper 接口与映射配置文件关联起来，系统可以通过调用自定义 Mapper 接口中的方法执行相应的 SQL 语句完成数据库操作，从而避免上述问题。

值得读者注意的是，开发人员无须编写自定义 Mapper 接口的实现，MyBatis 会自动为其创建动态代理对象。在有些场景中，自定义 Mapper 接口可以完全代替映射配置文件，但有的映射规则和 SQL 语句的定义还是写在映射配置文件中比较方便，例如动态 SQL 语句的定义。

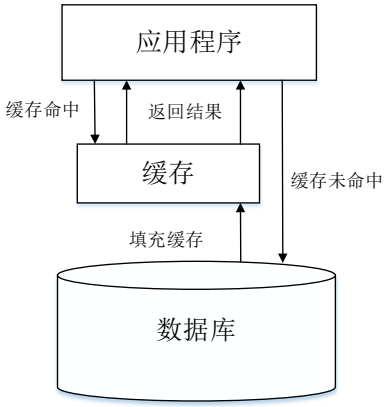

图 1-4

1.4.2 核心处理层

介绍完 MyBatis 的基础支持层之后，我们来分析 MyBatis 的核心处理层。在核心处理层中实现了 MyBatis 的核心处理流程，其中包括 MyBatis 的初始化以及完成一次数据库操作的涉及的全部流程。下面简单描述各个模块的功能，在第 3 章将会详细分析核心处理层的实现原理。

- **配置解析**

  在 MyBatis 初始化过程中，会加载 mybatis-config.xml 配置文件、映射配置文件以及 Mapper 接口中的注解信息，解析后的配置信息会形成相应的对象并保存到 Configuration 对象中。例如，示例中定义的<resultMap>节点（即 ResultSet 的映射规则）会被解析成 ResultMap 对象；示例中定义的<result>节点（即属性映射）会被解析成 ResultMapping 对象。之后，利用该 Configuration 对象创建 SqlSessionFactory 对象。

  待 MyBatis 初始化之后，开发人员可以通过初始化得到 SqlSessionFactory 创建 SqlSession 对象并完成数据库操作。

- **SQL 解析与 scripting 模块**

  拼凑 SQL 语句是一件烦琐且易出错的过程，为了将开发人员从这项枯燥无趣的工作中解脱出来，MyBatis 实现动态 SQL 语句的功能，提供了多种动态 SQL 语句对应的节点，例如，<where>节点、<if>节点、<foreach>节点等。通过这些节点的组合使用，开发人员可以写出几乎满足所有需求的动态 SQL 语句。

  MyBatis 中的 scripting 模块会根据用户传入的实参，解析映射文件中定义的动态 SQL 节点，并形成数据库可执行的 SQL 语句。之后会处理 SQL 语句中的占位符，绑定用户传入的实参。

- **SQL 执行**

  SQL 语句的执行涉及多个组件，其中比较重要的是 Executor、StatementHandler、ParameterHandler 和 ResultSetHandler。Executor 主要负责维护一级缓存和二级缓存，并提供事务管理的相关操作，它会将数据库相关操作委托给 StatementHandler 完成。

  StatementHandler 首先通过 ParameterHandler 完成 SQL 语句的实参绑定，然后通过 java.sql.Statement 对象执行 SQL 语句并得到结果集，最后通过 ResultSetHandler 完成结果集的映射，得到结果对象并返回。图 1-5 展示了 MyBatis 执行一条 SQL 语句的大致过程。

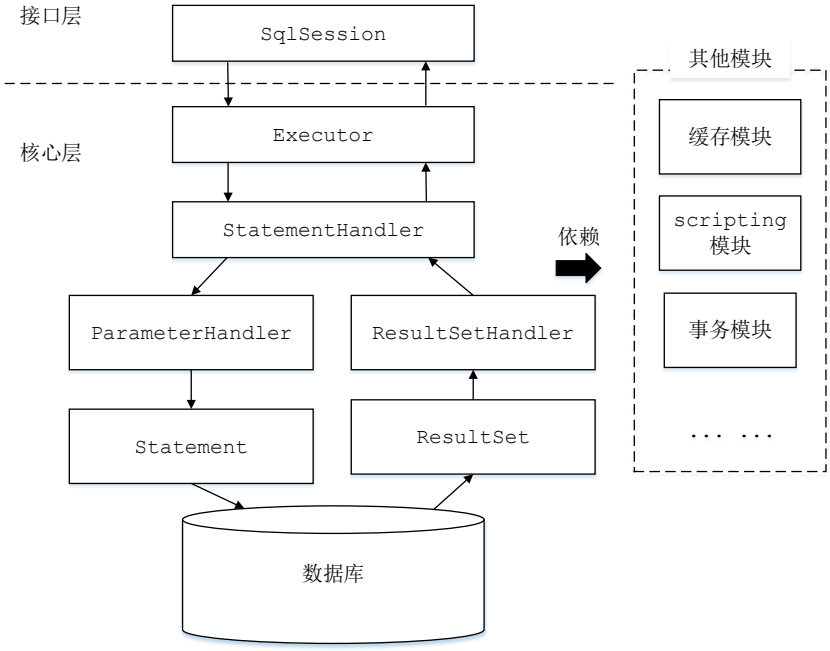

图 1-5

- **插件**

  Mybatis 自身的功能虽然强大，但是并不能完美切合所有的应用场景，因此 MyBatis 提供了插件接口，我们可以通过添加用户自定义插件的方式对 MyBatis 进行扩展。用户自定义插件也可以改变 Mybatis 的默认行为，例如，我们可以拦截 SQL 语句并对其进行重写。由于用户自定义插件会影响 MyBatis 的核心行为，在使用自定义插件之前，开发人员需要了解 MyBatis 内部的原理，这样才能编写出安全、高效的插件。

1.4.3 接口层

接口层相对简单，其核心是 SqlSession 接口，该接口中定义了 MyBatis 暴露给应用程序调用的 API，也就是上层应用与 MyBatis 交互的桥梁。接口层在接收到调用请求时，会调用核心处理层的相应模块来完成具体的数据库操作。SqlSession 接口及其具体实现将在第 3 章介绍。

1.5 本章小结

本章首先介绍了 ORM 框架出现的背景、意义以及相关概念。然后介绍了 Hibernate、JPA、Spring JDBC、MyBatis 这些常见持久化框架的优缺点，希望读者在进行技术选型时能有所参考。之后通过一个简单的示例，介绍了 MyBatis 中的 mybatis-config.xml 配置文件、映射配置文件中的核心配置，以及 MyBatis 的使用方式，帮助读者快速熟悉 MyBatis。最后我们介绍了 MyBatis 的整体架构，并简单介绍了 MyBatis 的基础支持层、核心处理层以及接口层中的主要模块的功能。