

书名：  
《深入浅出MyBatis技术原理与实战》  
出版时间：2016年9月

如需要本书的完整P D F版 请联系QQ:996727924 谢谢！！  
感谢群主和群管理员提供平台，祝您们身体健康，步步高升！

手把手教你整合开发MyBatis-Spring项目

# 深入浅出 MyBatis 技术原理与实战

杨开振 / 著

基于官方API的完全解读，开MyBatis应用之先河  
详细阐述MyBatis内部运行原理和插件开发

## 杨开振 ►

- 长期从事Java开发工作，拥有近十年的Java开发经验，目前就职于一家互联网金融公司，担任互联网软件开发职位。
- IT技术的狂热爱好者，热衷于Java互联网方向的软件技术开发与研究。
- 熟练掌握Java基础、软件开发设计模式和数据库相关知识，对Spring、MyBatis等主流Java开源框架有深入研究。





# 深入浅出 **MyBatis** 技术原理与实战

杨开振 / 著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING



## 内 容 提 要

随着大数据时代的到来,Java 持久层框架 MyBatis 已经成为越来越多企业的选择。遗憾的是,时至今日国内依然没有一本讨论 MyBatis 的书,这增加了初学者的学习难度,初学者往往只能基于零星的案例来学习 MyBatis,无法系统地掌握 MyBatis,更不用说精通了。《深入浅出 MyBatis 技术原理与实战》是笔者通过大量实践和研究源码后创作而成的,是国内第一本系统介绍 MyBatis 的著作。

本书分为 3 个部分,依次介绍了 MyBatis 的基础应用、原理及插件开发、实践应用,使读者能够由浅入深、循序渐进地掌握 MyBatis 技术。首先,本书在官方 API 的基础上完善了许多重要的论述和实例,并且给出了实操建议,帮助读者正确掌握 MyBatis。其次,本书详细讲述了 MyBatis 的内部运行原理,并全面讨论了插件的开发。最后,本着学以致用原则,笔者阐述了 MyBatis-Spring 项目和一些 MyBatis 开发常见的实例,使读者能够学得会,用得好。

本书不是一本味同嚼蜡的理论专著,而是一本 MyBatis 的实践指南,无论你是 Java 程序员、MyBatis 开发者,还是 Java 持久层框架的研究者,你都能从本书中收获知识。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有,侵权必究。

## 图书在版编目(CIP)数据

深入浅出 MyBatis 技术原理与实战 / 杨开振著. —北京: 电子工业出版社, 2016.9  
ISBN 978-7-121-29594-2

I. ①深… II. ①杨… III. ①JAVA 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字 (2016) 第 183295 号

责任编辑: 徐津平

印 刷: 中国电影出版社印刷厂

装 订: 三河市华成印务有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编: 100036

开 本: 787×980 1/16 印张: 16.75 字数: 310 千字

版 次: 2016 年 9 月第 1 版

印 次: 2016 年 9 月第 1 次印刷

定 价: 69.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888, 88258888。

质量投诉请发邮件至 [zltz@phei.com.cn](mailto:zltz@phei.com.cn), 盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

本书咨询联系方式: (010) 51260888-819, [faq@phei.com.cn](mailto:faq@phei.com.cn)。



# 前 言

随着手机、平板电脑等移动终端的广泛应用，移动互联网时代已经到来。在这个时代里，构建一个高效的平台并提供服务是移动互联网的基础，在众多的网站服务中，使用 Java 构建网站的不在少数。移动互联网的特点是大数据、高并发，对服务器往往要求分布式、高性能、高灵活等，而传统模式的 Java 数据库编程框架已经不再适用了。在这样的背景下，一个 Java 的持久框架 MyBatis 走入了我们的世界，它以封装少、高性能、可优化、维护简易等优点成为了目前 Java 移动互联网网站服务的首选持久框架，它特别适合分布式和大数据网络数据库的编程。

本书主要讲解了 MyBatis 的应用。从目前的情况来看，国内图书市场上没有介绍 MyBatis 的书籍，有的只是官方的 API 和少数的几篇博客文章，国外图书市场上的这类书籍也是凤毛麟角，这使得系统学习 MyBatis 困难重重。官方的 API 只是简单介绍了 MyBatis 有些什么功能和一些基本的使用方法，没有告诉我们如何用好，其中原理是什么，需要注意哪些问题，这显然是不够的。有些博客虽然讲解得比较深入，但是内容支离破碎，没有形成一个完整的知识体系，不易于初学者对 MyBatis 进行系统学习。随着移动互联网应用的兴起，系统掌握 MyBatis 编程技巧已经成了用 Java 构建移动互联网网站的必要条件。为了顺应时代的要求，笔者写下了这本书，以期为广大需要掌握 MyBatis 的开发者提供学习和参考的资料。

阅读本书要求开发人员拥有 Java 语言基础和 JDBC 基础知识，对数据库也要掌握入门知识，最好能够掌握常用的设计模式，因为在介绍 MyBatis 构造时，常常涉及设计模式，尤其是第 6 章和第 7 章的内容。

本书以讲解 MyBatis 基础运用和原理为主，所以适合初级到中级开发人员阅读。



本书分为三大部分。

第一部分是 MyBatis 基础应用，主要介绍如何高效地使用 MyBatis。

第 1 章：MyBatis 的内容简介，告诉读者 MyBatis 是什么，在何种场景下使用它。

第 2 章：主要介绍 MyBatis 的基础模块及其生命周期，并给出实例。

第 3 章：主要介绍 MyBatis 配置的主要含义和内容。

第 4 章：介绍 MyBatis 映射器的主要元素及其使用方法。

第 5 章：介绍动态 SQL，助你轻松应对大部分的 SQL 场景。

第二部分是 MyBatis 原理，我们将深入源码去理解 MyBatis 的内部运行原理以及插件的开发方法和技巧。

第 6 章：介绍 MyBatis 的解析和运行原理，我们将了解到 SqlSession 的构建方法，以及其四大对象是如何工作的。

第 7 章：在第 6 章的基础上着重介绍 MyBatis 的插件，这里我们将学习插件的设计原理，以及开发方法和注意的要点。

第三部分是 MyBatis 的实战应用，主要讲解 MyBatis 的一些实用的场景。

第 8 章：介绍 MyBatis-Spring，主要讲解如何在 Spring 项目中集成 MyBatis 应用，帮助读者在 Spring 的环境中顺利使用 MyBatis。

第 9 章：介绍 MyBatis 的实用场景，精选一批典型且又常用的场景。详细解析每一个场景下，开发人员需要注意避免的一些错误和性能上的损失。

MyBatis 源于 2002 年的 iBatis 项目，至今 MyBatis 中依然有许多 iBatis 的痕迹。本书默认使用 MyBatis 的版本是 3.3.0，使用 MyBatis-Spring 的版本是 1.2.3。而历史上的 iBatis 的书籍已经跟不上技术发展的步伐，于是笔者通过自己的努力和实践，在研究 MyBatis 源码的基础上，写作本书。从本书中既能学习如何使用 MyBatis，也可以学习 MyBatis 的原理和应用，为国内的 MyBatis 开发者提供一条系统掌握 MyBatis 编程技巧的捷径，当然读者也可以把本书作为工具书参考。在实际操作中，MyBatis 往往是结合 Spring 使用的，于是本书花费了一些篇幅讲解 MyBatis-Spring 技术，笔者也会略略提到 Spring 项目的内容，以便更好地论述它们。最后笔者还将讲解一些使用频率高、参考价值大的场景，使读者能熟练掌握 MyBatis 的开发。



本书坚持实用原则，对于一些使用频率低的技术并没有提及太多，比如注解 SQL、SQL 构造器等内容，使用这些内容，会造成代码的可读性下降。

感谢我的公司为我提供真实的使用 MyBatis 的环境，所有的程序代码都经过了调试。感谢我的姐姐杨坚，她参与编写并通篇审校了本书，润色了那些晦涩的句子。同时也感谢电子工业出版社的编辑们，尤其是汪达文的全程跟进。没有他们的辛苦付出，就没有本书的成功出版。在出版本书的欣喜之余，也伴着战战兢兢，因为笔者才疏学浅，很多东西都是从对源码的理解和实际操作中获得的，因此书中难免有疏漏之处，或有不能让读者满意的地方。如果有困惑，读者可以发邮件到我的邮箱：ykzhen2013@163.com，也可以在我的博客（<http://blog.csdn.net/ykzhen2015>）中和我讨论，还望各位同行不吝赐教。

杨开振

2016 年 7 月

# 目 录

第 1 章	MyBatis 简介 .....	1
1.1	传统的 JDBC 编程 .....	1
1.2	ORM 模型 .....	4
1.3	Hibernate .....	4
1.4	MyBatis .....	9
1.5	什么时候用 MyBatis .....	12
第 2 章	MyBatis 入门 .....	13
2.1	开发环境准备 .....	13
2.1.1	下载 MyBatis .....	13
2.1.2	搭建开发环境 .....	14
2.2	MyBatis 的基本构成 .....	15
2.2.1	构建 SqlSessionFactory .....	15
2.2.2	创建 SqlSession .....	19
2.2.3	映射器 .....	21
2.3	生命周期 .....	26
2.3.1	SqlSessionFactoryBuilder .....	27
2.3.2	SqlSessionFactory .....	27
2.3.3	SqlSession .....	27
2.3.4	Mapper .....	28
2.4	实例 .....	28



第3章 配置	37
3.1 properties 元素	38
3.1.1 property 子元素	38
3.1.2 properties 配置文件	39
3.1.3 程序参数传递	39
3.1.4 优先级	40
3.2 设置	41
3.3 别名	44
3.3.1 系统定义别名	44
3.3.2 自定义别名	47
3.4 typeHandler 类型处理器	48
3.4.1 系统定义的 typeHandler	49
3.4.2 自定义 typeHandler	51
3.4.3 枚举类型 typeHandler	55
3.5 ObjectFactory	62
3.6 插件	65
3.7 environments 配置环境	65
3.7.1 概述	65
3.7.2 数据库事务	66
3.7.3 数据源	67
3.8 databaseIdProvider 数据库厂商标识	68
3.8.1 使用系统默认规则	68
3.8.2 不使用系统默认规则	69
3.9 引入映射器的方法	71
第4章 映射器	73
4.1 映射器的主要元素	73
4.2 select 元素	74
4.2.1 概述	74
4.2.2 简易数据类型的例子	75
4.2.3 自动映射	76



4.2.4 传递多个参数 .....	78
4.2.5 使用 resultMap 映射结果集 .....	81
4.3 insert 元素 .....	82
4.3.1 概述 .....	82
4.3.2 主键回填和自定义 .....	83
4.4 update 元素和 delete 元素 .....	85
4.5 参数 .....	85
4.5.1 参数配置 .....	86
4.5.2 存储过程支持 .....	86
4.5.3 特殊字符串替换和处理（#和\$） .....	87
4.6 sql 元素 .....	88
4.7 resultMap 结果映射集 .....	89
4.7.1 resultMap 元素的构成 .....	89
4.7.2 使用 map 存储结果集 .....	91
4.7.3 使用 POJO 存储结果集 .....	91
4.7.4 级联 .....	92
4.8 缓存 cache .....	113
4.8.1 系统缓存（一级缓存和二级缓存） .....	113
4.8.2 自定义缓存 .....	117
第 5 章 动态 SQL .....	119
5.1 概述 .....	119
5.2 if 元素 .....	120
5.3 choose、when、otherwise 元素 .....	120
5.4 trim、where、set 元素 .....	121
5.5 foreach 元素 .....	123
5.6 test 的属性 .....	124
5.7 bind 元素 .....	125
第 6 章 MyBatis 的解析和运行原理 .....	127
6.1 涉及的技术难点简介 .....	128
6.1.1 反射技术 .....	129

6.1.2	JDK 动态代理	130
6.1.3	CGLIB 动态代理	133
6.2	构建 SqlSessionFactory 过程	134
6.2.1	构建 Configuration	135
6.2.2	映射器的内部组成	136
6.2.3	构建 SqlSessionFactory	138
6.3	SqlSession 运行过程	138
6.3.1	映射器的动态代理	138
6.3.2	SqlSession 下的四大对象	142
6.3.3	SqlSession 运行总结	150
第 7 章	插件	152
7.1	插件接口	152
7.2	插件的初始化	153
7.3	插件的代理和反射设计	154
7.4	常用的工具类——MetaObject	157
7.5	插件开发过程和实例	159
7.5.1	确定需要拦截的签名	159
7.5.2	实现拦截方法	161
7.5.3	配置和运行	162
7.5.4	插件实例	163
7.6	总结	166
第 8 章	MyBatis-Spring	168
8.1	Spring 的基础知识	168
8.1.1	Spring IOC 基础	169
8.1.2	Spring AOP 基础	171
8.1.3	Spring 事务管理	173
8.1.4	Spring MVC 基础	179
8.2	MyBatis-Spring 应用	181
8.2.1	概述	181
8.2.2	配置 SqlSessionFactory	182



8.2.3	配置 SqlSessionTemplate .....	184
8.2.4	配置 Mapper .....	188
8.2.5	配置事务 .....	190
8.3	实例 .....	191
8.3.1	环境准备 .....	191
8.3.2	文件目录 .....	193
8.3.3	Spring 配置文件 .....	194
8.3.4	MyBatis 框架相关配置 .....	198
8.3.5	配置服务层 .....	205
8.3.6	编写控制器 .....	209
8.3.7	测试 .....	210
8.4	总结 .....	210
第 9 章	实用的场景 .....	212
9.1	数据库 BLOB 字段读写 .....	212
9.2	批量更新 .....	215
9.3	调用存储过程 .....	217
9.3.1	存储过程 in 和 out 参数的使用 .....	217
9.3.2	存储过程游标 .....	220
9.4	分表 .....	225
9.5	分页 .....	227
9.5.1	RowBounds 分页 .....	227
9.5.2	插件分页 .....	229
9.6	上传文件到服务器 .....	239
9.7	在映射中使用枚举 .....	247
9.8	多对多级联 .....	249
9.9	总结 .....	253
附录 A	数据库模型描述与级联学生关系建表语句 .....	254



# 第 1 章

## MyBatis 简介

本章主要介绍了 Java ORM 的来源和历史,同时分别介绍了 JDBC、Hibernate 和 MyBatis 三种访问数据库的方法,在分析它们优缺点的基础上,比较它们之间的区别和适用的场景。

### 1.1 传统的 JDBC 编程

Java 程序都是通过 JDBC (Java Data Base Connectivity) 连接数据库的,这样我们就可以通过 SQL 对数据库编程。JDBC 是由 SUN 公司 (SUN 公司后被 Oracle 公司收购) 提出的一系列规范,但是它只定义了接口规范,而具体的实现是交由各个数据库厂商去实现的,因为每个数据库都有其特殊性,这些是 Java 规范没有办法确定的,所以 JDBC 就是一种典型的桥接模式。

传统的 JDBC 编程的使用给我们带来了连接数据库的功能,但是也引发了巨大的问题。代码清单 1-1 是用 JDBC 编程的一个例子。我们将从 MySQL 数据库中查询一个角色的名称,假设我们已经知道角色编号为 1。

代码清单 1-1: JdbcExample.java

---

```
public class JdbcExample {  
    private Connection getConnection() {  
        Connection connection = null;  
        try {  
            Class.forName("com.mysql.jdbc.Driver");  
            String url = "jdbc:mysql://localhost:3306/mybatis?zeroDateTime  
Behavior=convertToNull";  
            String user = "root";
```

```
String password = "learn";
connection = DriverManager.getConnection(url, user, password);
} catch (ClassNotFoundException | SQLException ex) {
    Logger.getLogger(JdbcExample.class.getName()).log(Level.SEVERE,
null, ex);
    return null;
}
return connection;
}

public Role getRole(Long id) {
    Connection connection = getConnection();
    PreparedStatement ps = null;
    ResultSet rs = null;
    try {
        ps = connection.prepareStatement("select id, role_name, note from
t_role where id = ?");
        ps.setLong(1, id);
        rs = ps.executeQuery();
        while(rs.next()) {
            Long roleId = rs.getLong("id");
            String userName = rs.getString("role_name");
            String note = rs.getString("note");
            Role role = new Role();
            role.setId(id);
            role.setRoleName(userName);
            role.setNote(note);
            return role;
        }
    } catch (SQLException ex) {
        Logger.getLogger(JdbcExample.class.getName()).log(Level.SEVERE,
null, ex);
    } finally {
        this.close(rs, ps, connection);
    }
    return null;
}

private void close(ResultSet rs, Statement stmt, Connection connection)
{
    try {
        if (rs != null && !rs.isClosed()) {
            rs.close();
        }
    }
```



```

    }
    } catch (SQLException ex) {
        Logger.getLogger(JdbcExample.class.getName()).log(Level.SEVERE,
null, ex);
    }
    try {
        if (stmt != null && !stmt.isClosed()) {
            stmt.close();
        }
    } catch (SQLException ex) {
        Logger.getLogger(JdbcExample.class.getName()).log(Level.SEVERE,
null, ex);
    }
    try {
        if (connection != null && !connection.isClosed()) {
            connection.close();
        }
    } catch (SQLException ex) {
        Logger.getLogger(JdbcExample.class.getName()).log(Level.SEVERE,
null, ex);
    }
}

public static void main(String[] args) {
    JdbcExample example = new JdbcExample();
    Role role = example.getRole(1L);
    System.err.println("role_name => " + role.getRoleName());
}
}

```

从代码中我们可以看出整个过程大致分为以下几步：

- 使用 JDBC 编程需要连接数据库，注册驱动和数据库信息。
- 操作 Connection，打开 Statement 对象。
- 通过 Statement 执行 SQL，返回结果到 ResultSet 对象。
- 使用 ResultSet 读取数据，然后通过代码转化为具体的 POJO 对象。
- 关闭数据库相关资源。

使用传统的 JDBC 方式存在一些弊端。其一，工作量相对较大。我们需要先连接，然后处理 JDBC 底层事务，处理数据类型。我们还需要操作 Connection 对象、Statement 对象



和 `ResultSet` 对象去拿到数据，并准确关闭它们。其二，我们要对 JDBC 编程可能产生的异常进行捕捉处理并正确关闭资源。对于一个简单的 SQL 在 JDBC 中尚且如此复杂，何况是更为复杂的应用呢？很快这种模式就被一些新的方法取代，于是 ORM 模型就出现了。不过所有的 ORM 模型都是基于 JDBC 进行封装的，不同的 ORM 模型对 JDBC 封装的强度是不一样的。

## 1.2 ORM 模型

由于 JDBC 存在的缺陷，在实际工作中我们很少使用 JDBC 进行编程，于是提出了对象关系映射（Object Relational Mapping，简称 ORM，或者 O/RM，或者 O/R mapping）。那什么是 ORM 模型呢？

简单地说，ORM 模型就是数据库的表和简单 Java 对象（Plain Ordinary Java Object，简称 POJO）的映射关系模型，它主要解决数据库数据和 POJO 对象的相互映射。我们通过这层映射关系就可以简单迅速地把数据库表的数据转化为 POJO，以便程序员更加容易理解和应用 Java 程序，如图 1-1 所示。

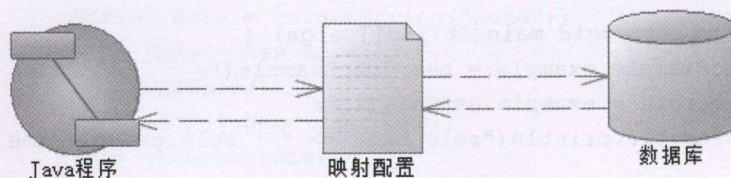


图 1-1 ORM 映射模型

有了 ORM 模型，在大部分情况下，程序员只需要了解 Java 应用而无需对数据库相关知识深入了解，便可以写出通俗易懂的程序。此外，ORM 模型提供了统一的规则使得数据库的数据通过配置便可轻易映射到 POJO 上。

## 1.3 Hibernate

最初 SUN 公司推出了 Java EE 服务器端组件模型（EJB），但是由于 EJB 配置复杂，且适用范围较小，于是很快就被淘汰了。与 EJB 的失败伴随而来的是另外一个框架的应运而



生。它就是从诞生至今都十分流行的 Hibernate。

Hibernate 一问世就成了 Java 世界首选的 ORM 模型，它是建立在 POJO 和数据库表模型的直接映射关系上的。

Hibernate 是建立在若干 POJO 通过 XML 映射文件（或注解）提供的规则映射到数据库表上的。换句话说，我们可以通过 POJO 直接操作数据库的数据。它提供的是一种全表映射的模型。如图 1-2 所示是 Hibernate 模型的开发过程。相对而言，Hibernate 对 JDBC 的封装程度还是比较高的，我们已经不需要编写 SQL 语言（Structured Query Language），只要使用 HQL 语言（Hibernate Query Language）就可以了。

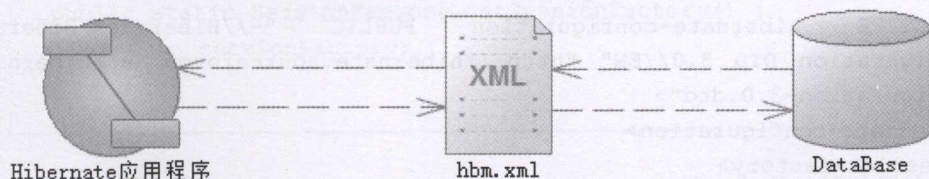


图 1-2 Hibernate 模型的开发过程

首先我们需要提供 hbm.xml 文件，制定映射规则。下面以开发角色类为例进行讲解，如代码清单 1-2 所示。

代码清单 1-2: TRole.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<!-- Generated 2015-12-12 22:50:58 by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
    <class name="com.learn.mybatis.chapter1.po.TRole" table="t_role"
catalog="mybatis" optimistic-lock="version">
        <id name="id" type="long">
            <column name="id" />
            <generator class="assigned" />
        </id>
        <property name="roleName" type="string">
            <column name="role_name" length="60" />
        </property>
        <property name="note" type="string">
            <column name="note" length="512" />
        </property>
    </class>
</hibernate-mapping>
```



```
        </property>
    </class>
</hibernate-mapping>
```

---

这是一个简单的 XML 文件，它描述的是 POJO 和数据库表的映射关系。Hibernate 通过配置文件(或注解)就可以把数据库的数据直接映射到 POJO 上，我们可以通过操作 POJO 去操作数据库记录。对于不擅长 SQL 的程序员来说，这是莫大的惊喜，因为通过 Hibernate 你几乎不需要编写 SQL 就能操作数据库的记录。代码清单 1-3 是 Hibernate 的配置信息。

代码清单 1-3: hibernate.cfg.xml

---

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property
>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/mybatis?zero
DateTimeBehavior=convertToNull</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">learn</property>
        <mapping resource="com/learn/mybatis/chapter1/po/TUser.hbm.xml"/>
        <mapping resource="com/learn/mybatis/chapter1/po/TRole.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

---

别看代码很多，但是全局就是这样的一个 XML 文件，作为数据库连接信息，配置信息也相对简易。然后建立 Hibernate 的工厂对象 (SessionFactory)，用它来做全局对象，产生 Session 接口，就可以操作数据库了，如代码清单 1-4 所示。

代码清单 1-4: HibernateUtil

---

```
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
```

---

```

    try {
        Configuration    cfg    =    new    Configuration().configure
("hibernate.cfg.xml");
        sessionFactory = cfg.buildSessionFactory();
    } catch (Throwable ex) {
        System.err.println("Initial SessionFactory creation failed." +
ex);
        throw new ExceptionInInitializerError(ex);
    }
}

public static SessionFactory getSessionFactory() {
    return sessionFactory;
}
}

```

---

上面的操作为的是产生 Hibernate 的 SessionFactory。它作为全局，可以到处引用，那么剩下的使用就非常简单了。比如我们可以用它来实现代码清单 1-1 的功能，如代码清单 1-5 所示。

代码清单 1-5: HibernateExample.java

---

```

public class HibernateExample {
    public static void main(String[] args) {
        Session session = null;
        try {
            session = HibernateUtil.getSessionFactory().openSession();
            TRole role = (TRole)session.get(TRole.class, 1L);
            System.err.println("role_name = >" + role.getRoleName());
        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
}

```

---

按照代码清单 1-5 的方法来实现代码清单 1-1 的功能有以下好处：

- 消除了代码的映射规则，它全部被分离到了 XML 或者注解里面去配置。
- 无需再管理数据库连接，它也配置在 XML 里面。



- 一个会话中，不要操作多个对象，只要操作 Session 对象即可。
- 关闭资源只需要关闭一个 Session 便可。

这就是 Hibernate 的优势，在配置了映射文件和数据库连接文件后，Hibernate 就可以通过 Session 操作，非常容易，消除了 JDBC 带来的大量代码，大大提高了编程的简易性和可读性。此外，它还提供级联、缓存、映射、一对多等功能，以便我们使用。正因为具有这些优势，Hibernate 成为了时代的主流框架，被大量应用在各种 Java 数据库的访问中。Hibernate 是全表映射，你可以通过 HQL 去操作 POJO 进而操作数据库的数据。

但是 Hibernate 有缺陷吗？当然有，世界上没有完美无缺的方案。作为全表映射框架，举个例子来说，如果有张账务表（按年分表），比如 2015 年表命名为 bill2015，到了 2016 年表命名为 bill2016，要动态加映射关系，Hibernate 需要破坏底层封装才能做到。又比如说，一些账务信息往往需要和某些对象关联起来，不同的对象有不同的列，因此列名也是无法确定的，显然我们没有办法配置 XML 去完成映射规则。再者如果使用存储过程，Hibernate 也是无法适应的。这些都不是致命的，最为致命的问题是性能。Hibernate 屏蔽了 SQL，那就意味着只能全表映射，但是一张表可能有几十到上百个字段，而你感兴趣的只有 2 个，这是 Hibernate 无法适应的。尤其是在大型网站系统，对传输数据有严格规定，不能浪费带宽的场景下就更为明显了。有很复杂的场景需要关联多张表，Hibernate 全表逐级取对象的方法也只能作罢，写 SQL 还需要手工的映射取数据，这带来了很大的麻烦。此外，如果我们需要优化 SQL，Hibernate 也是无法做到的。

我们稍微总结一下 Hibernate 的缺点：

- 全表映射带来的不便，比如更新时需要发送所有的字段。
- 无法根据不同的条件组装不同的 SQL。
- 对多表关联和复杂 SQL 查询支持较差，需要自己写 SQL，返回后，需要自己将数据组装为 POJO。
- 不能有效支持存储过程。
- 虽然有 HQL，但是性能较差。大型互联网系统往往需要优化 SQL，而 Hibernate 做不到。

在当今大型互联网中，灵活、SQL 优化，减少数据的传递是最基本的优化方法，显然 Hibernate 无法满足我们的要求。这时 MyBatis 框架诞生了，它提供了更灵活、更方便的方法，弥补了 Hibernate 的这些缺陷。



## 1.4 MyBatis

为了解决 Hibernate 的不足，一个半自动映射的框架 MyBatis 应运而生。之所以称它为半自动，是因为它需要手工匹配提供 POJO、SQL 和映射关系，而全表映射的 Hibernate 只需要提供 POJO 和映射关系便可。

历史上，MyBatis 的前身是 Apache 的一个开源项目 iBatis，2010 年这个项目由 apache software foundation 迁移到了 google code，并且改名为 MyBatis。2013 年 11 月迁移到 Github，所以目前 MyBatis 是由 Github 维护的。

iBatis 一词来源于“internet”和“abatis”的组合，是一个基于 Java 的持久层框架。iBatis 提供的持久层框架包括 SQL Maps 和 DAO(Data Access Objects)。它能很好地解决 Hibernate 遇到的问题。与 Hibernate 不同的是，它不单单要我们提供映射文件，还需要我们提供 SQL 语句。MyBatis 所需要提供的映射文件包含以下三个部分。

- SQL。
- 映射规则。
- POJO。

在 MyBatis 里面，你需要自己编写 SQL，虽然比 Hibernate 配置得多，但是 MyBatis 可以配置动态 SQL，这就解决了 Hibernate 的表名根据时间变化，不同的条件下列名不一样的问题。同时你也可以优化 SQL，通过配置决定你的 SQL 映射规则，也能支持存储过程，所以对于一些复杂的和需要优化性能 SQL 的查询它更加方便，MyBatis 几乎能做到 JDBC 所能做到的所有事情。MyBatis 具有自动映射功能。换句话说，在注意一些规则的基础上，MyBatis 可以给我们完成自动映射，而无需再写任何的映射规则，这大大提高了开发效率和灵活性。

如图 1-3 所示为 MyBatis 的 ORM 映射模型。

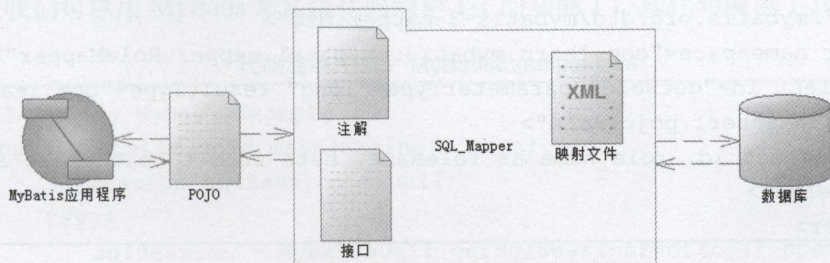


图 1-3 MyBatis 的 ORM 映射模型



让我们看看如何实现 `JdbcExample` 的功能。首先是数据库及其他的基础配置，如代码清单 1-6 所示。

代码清单 1-6: mybatis\_config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/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/
mybatis"/>
                <property name="username" value="root"/>
                <property name="password" value="learn"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="com\learn\mybatis\chapter1\pojo\role.xml" />
    </mappers>
</configuration>
```

---

这就是 MyBatis 的基础配置文件。其次是一个映射文件，也十分简单，如代码清单 1-7 所示。

代码清单 1-7: Role.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.learn.mybatis.chapter1.mapper.RoleMapper">
    <select id="getRole" parameterType="long" resultType="com.learn.
mybatis.chapter1.pojo.Role">
        select id, role_name as roleName, note from t_role where id=#{id}
    </select>
</mapper>
```

---

这里我们给出了 SQL，但是并没有给出映射规则，因为这里我们使用的 SQL 列名和

POJO 的属性名保持一致，这个时候 MyBatis 会自动提供映射规则，所以省去了这部分的配置工作。再者，我们还需要一个接口，注意仅仅是接口，而无需实现类，如代码清单 1-8 所示。

代码清单 1-8: RoleMapper.java

---

```
public interface RoleMapper {
    public Role getRole(Long id);
}
```

---

为了使用 MyBatis，我们还需要建立 SqlSessionFactory，如代码清单 1-9 所示。

代码清单 1-9: MyBatisUtil.java

---

```
public class MyBatisUtil {
    private static SqlSessionFactory sqlSessionFactory = null;
    public static SqlSessionFactory getSqlSessionFactroy() {
        InputStream inputStream = null;
        if (sqlSessionFactory == null) {
            try {
                String resource = "mybatis_config.xml";
                sqlSessionFactory = new SqlSessionFactoryBuilder().build
                    (Resources.getResourceAsStream(resource));
                return sqlSessionFactory;
            } catch (Exception ex) {
                System.err.println(ex.getMessage());
                ex.printStackTrace();
            }
        }
        return sqlSessionFactory;
    }
}
```

---

现在我们可以用 MyBatis 来实现代码清单 1-1 的功能了，如代码清单 1-10 所示。

代码清单 1-10: MyBatisExample.java

---

```
public class MyBatisExample {
    public static void main(String[] args) {
        SqlSession sqlSession = null;
        try {
            sqlSession = MyBatisUtil.getSqlSessionFactroy().openSession();
            RoleMapper roleMapper = sqlSession.getMapper(RoleMapper.class);
        }
```

---



```
        Role role = roleMapper.getRole(1L);
        System.err.println("role_name = >" + role.getRoleName());
    } finally {
        sqlSession.close();
    }
}
}
```

---

这样便完成了 MyBatis 的代码编写工作，SQL 和映射规则都在 XML 里面进行了分离，而 MyBatis 更为灵活。你可以自由书写 SQL，定义映射规则。此外，MyBatis 提供接口编程的映射器只需要一个接口和映射文件便可以运行，消除了在 iBatis 时代需要 SqlSession 调度的情况。

## 1.5 什么时候用 MyBatis

通过对 JDBC、Hibernate 和 MyBatis 的介绍，我们有了一些认识。JDBC 的方式在目前而言极少用到，因为你需要提供太多的代码，操作太多的对象，麻烦不说，还极其容易出错，所以这不是一种推荐的方式，在实际开发中直接用 JDBC 的场景也是很少的。

Hibernate 作为较为流行的 Java ORM 框架，它确实编程简易，需要我们提供映射的规则，完全可以通过 IDE 生成，同时无需编写 SQL 确实开发效率优于 MyBatis。此外，它也提供了缓存、日志、级联等强大的功能，但是 Hibernate 的缺陷也是十分明显的，多表关联复杂 SQL，数据系统权限限制，根据条件变化的 SQL。存储过程等场景使用 Hibernate 十分不便，而性能又难以通过 SQL 优化。所以注定了 Hibernate 只适用于在场景不太复杂，要求性能不太苛刻的时候使用。

如果你需要一个灵活的、可以动态生成映射关系的框架，那么 MyBatis 确实是一个最好的选择。它几乎可以代替 JDBC，拥有动态列、动态表名，存储过程都支持，同时提供了简易的缓存、日志、级联。但是它的缺陷是需要你提供映射规则和 SQL，所以它的开发工作量比 Hibernate 略大一些。

你需要根据项目的实际情况去选择框架。因为 MyBatis 具有高度灵活、可优化、易维护等特点，所以它目前是大型移动互联网项目的首选框架。

下面各章，我们将分别讨论 MyBatis 的应用、原理和实践。



## 第2章

# MyBatis 入门

本章的目标很明确，就是带大家入门。我们先准备环境的搭建，然后开始讲述 MyBatis 的基本构成和应用，并且给出一个可以运行的实例。为了让大家加深理解，我们将讲述 MyBatis 的核心类和接口对象的生命周期，在理解其生命周期后，我们将优化实例。本章内容应用多于原理，我们在后面的几章中再讨论其实现的原理、架构和方法。

### 2.1 开发环境准备

学习编程是一门实践科学，只有一边编写代码一边学习才会有好的效果，所以需要搭建一个可以运行的学习环境，以便我们实践和探索，所以这节主要带领大家来配置开发环境。

#### 2.1.1 下载 MyBatis

输入网址 <https://github.com/mybatis/mybatis-3/releases> 进入 MyBatis 的官网，我们即可下载 MyBatis，如图 2-1 所示。

我们可以在这里下载到 MyBatis 所需的 jar 包和源码包。讲解 MyBatis 运行原理和插件的时候常常会用到源码的内容。

使用 MyBatis 项目可以参考 <http://mybatis.org/mybatis-3/zh/index.html>。

使用 MyBatis-Spring 项目可以参考 <http://mybatis.org/spring/zh/index.html>。



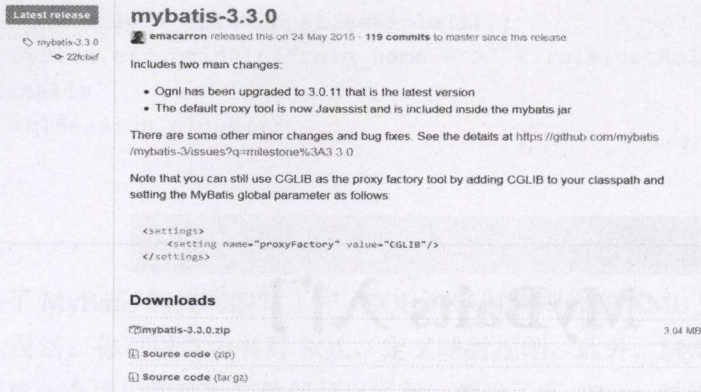


图 2-1 下载 MyBatis

## 2.1.2 搭建开发环境

无论使用哪一种 Java IDE 都可以轻松搭建开发环境。这里以 Eclipse 为例搭建我们的开发环境。我们打开下载得到的 MyBatis 开发包就可以得到如图 2-2 所示的目录。

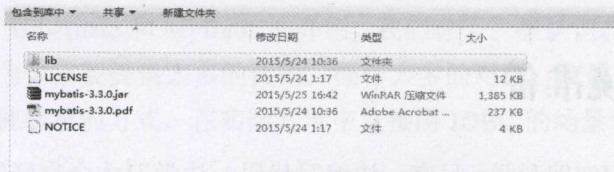


图 2-2 MyBatis 下载包目录

这里的 jar 文件分为两类，一类是 MyBatis 本身的 jar 包，另一类在 lib 文件夹里。MyBatis 项目所依赖的 jar 包，而 pdf 文件则是它提供的 API 文档。我们只需要在 Eclipse 中引入 MyBatis 的 jar 包即可，如图 2-3 所示。

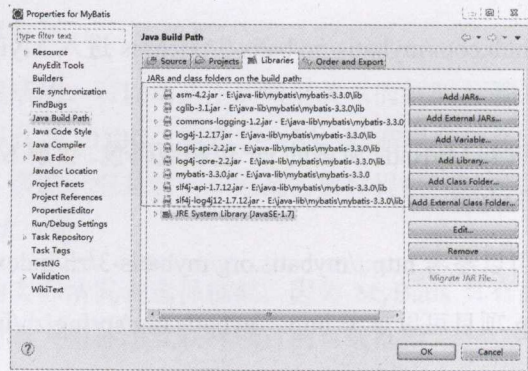


图 2-3 引入 MyBatis 的 jar 包

这样便完成了 MyBatis 的环境搭建，我们便可以在项目中使用 MyBatis 了。

## 2.2 MyBatis 的基本构成

认识往往是从表面现象到内在本质的一个探索过程，所以对于 MyBatis 的掌握，我们从认识“表面现象”——MyBatis 的基本构成开始。我们先了解一下 MyBatis 的核心组件。

- `SqlSessionFactoryBuilder`（构造器）：它会根据配置信息或者代码来生成 `SqlSessionFactory`（工厂接口）。
- `SqlSessionFactory`：依靠工厂来生成 `SqlSession`（会话）。
- `SqlSession`：是一个既可以发送 SQL 去执行并返回结果，也可以获取 `Mapper` 的接口。
- `SQL Mapper`：它是 MyBatis 新设计的组件，它是由一个 Java 接口和 XML 文件（或注解）构成的，需要给出对应的 SQL 和映射规则。它负责发送 SQL 去执行，并返回结果。

用一张图表达它们之间的关联，如图 2-4 所示。

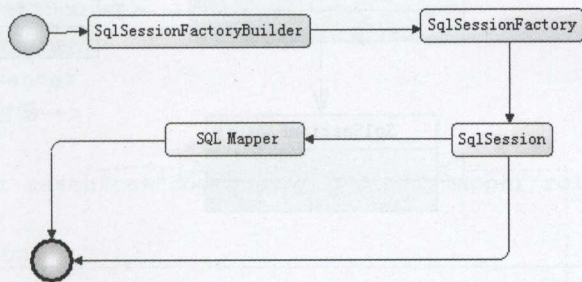


图 2-4 MyBatis 的构成

这里我们不需要马上明白 MyBatis 的组件内容，先了解它们之间的先后顺序、流程和基本功能，后面我们会详细讨论它们的用法。

### 2.2.1 构建 `SqlSessionFactory`

每个 MyBatis 的应用都是以 `SqlSessionFactory` 的实例为中心的。`SqlSessionFactory` 的实例可以通过 `SqlSessionFactoryBuilder` 获得。但是读者们需要注意 `SqlSessionFactory` 是一个



工厂接口而不是现实类，它的任务是创建 `SqlSession`。`SqlSession` 类似于一个 JDBC 的 `Connection` 对象。MyBatis 提供了两种模式去创建 `SqlSessionFactory`：一种是 XML 配置的方式，这是笔者推荐的方式；另一种是代码的方式。能够使用配置文件的时候，我们要尽量地使用配置文件，这样一方面可以避免硬编码（hard code），一方面方便日后配置人员的修改，避免重复编译代码。

这里我们的 `Configuration` 的类全限定名为 `org.apache.ibatis.session.Configuration`，它在 MyBatis 中将以一个 `Configuration` 类对象的形式存在，而这个对象将存在于整个 MyBatis 应用的生命期中，以便重复读取和运用。在内存中的数据是计算机系统中读取速度最快的，我们可以解析一次配置的 XML 文件保存到 `Configuration` 类对象中，方便我们从这个对象中读取配置信息，性能高。单例占用空间小，基本不占用存储空间，而且可以反复使用。`Configuration` 类对象保存着我们配置在 MyBatis 的信息。在 MyBatis 中提供了两个 `SqlSessionFactory` 的实现类，`DefaultSqlSessionFactory` 和 `SqlSessionManager`。不过 `SqlSessionManager` 目前还没有使用，MyBatis 中目前使用的是 `DefaultSqlSessionFactory`。

让我们看看它们的关系图，如图 2-5 所示。

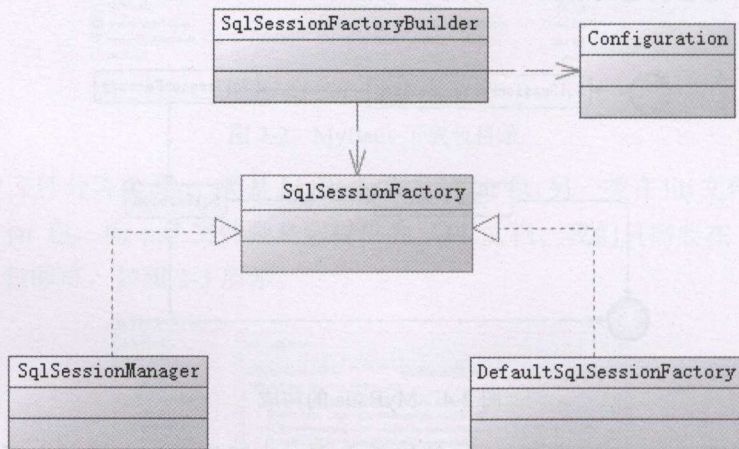


图 2-5 两个 `SqlSessionFactory` 现实类的关系图

### 2.2.1.1 使用 XML 方式构建

这里我们配置一个简易的 XML，包含获取数据库连接实例的数据源（`DataSource`）、决定事务范围和控制方式的事务管理器（`TransactionManager`）和映射器（`SQL Mapper`）。XML 配置文件的内容后面会详细探讨，这里先给出一个简单的示例，如代码清单 2-1 所示。



代码清单 2-1: mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <!--定义别名-->
  <typeAliases>
    <typeAlias alias="role" type="com.learn.chapter2.po.Role"/>
  </typeAliases>
  <!--定义数据库信息,默认使用 development 数据库构建环境-->
  <environments default="development">
    <environment id="development">
      <!--采用 jdbc 事务管理-->
      <transactionManager type="JDBC"/>
      <!--配置数据库链接信息-->
      <dataSource type="POOLED">
        <property name="driver" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/mybatis"/>
        <property name="username" value="root"/>
        <property name="password" value="learn"/>
      </dataSource>
    </environment>
  </environments>
  <!--定义映射器-->
  <mappers>
    <mapper resource="com/learn/chapter2/mapper/roleMapper.xml"/>
  </mappers>
</configuration>
```

对上面的配置做一下说明。

- 这里配置了一个别名 `role`，它代表 `com.learn.chapter2.po.Role`，这样我们就可以在 MyBatis 上下文中引用它了。
- 我们配置了环境内容，它默认使用 `id` 是 `development` 的环境配置，包含以下两方面的内容。

(1) 采用 JDBC 的事务管理模式。

(2) 数据库的连接信息。