



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Java持久化之MyBatis 3

[原名 : Java Persistence with MyBatis 3]

一本关于 简单但功能强大的Java持久化框架-MyBatis 的实用指导书 !

A practical guide to MyBatis,a simple yet powerful Java Persistence Framework !

K. Siva Prasad Reddy 著

娄 变

译

[PACKT] open source*
PUBLISHING community experience distilled

关于作者

K. Siva Prasad Reddy 是一名资深软件工程师,居住于印度海得拉巴市,拥有 6 年以上的 Java 和 JavaEE 技术企业级应用开发经验。Siva 是拥有 Sun 公司认证的 Java 程序员,有丰富的服务器端技术开发经验,如: Java,JavaEE,Spring, MyBatis, JSF (PrimeFaces) 和 WebServices (SOAP/REST)。

Siva 平时通过他的博客 www.sivalabs.in 分享他获得的知识。如果你想了解他更多的工作信息,你可以关注他的 Twitter (@sivalabs) 和 GitHub (<https://github.com/sivaprasadreddy>)。

“我要感谢我妻子 Neha,感谢她在我写本书过程的每一步的支持,如果没有她,本书也不可能完成。感谢我的父母和妹妹给我的精神支持,帮助我完成这个(书写本书的)梦想。”

关于本书评审员

Muhammad Edwin 是 Baculsoft Technology 的创建者和技术总监，Baculsoft Technology 公司印度尼西亚系统集成方面先驱，在开源技术方面提供技术咨询，支持和服务。他的主要职责是运用 Java 前沿企业级技术设计和实现解决方案来满足客户的需求。他身兼多职，包括软件工程师、开发团队 Leader、Java 培训师。Edwin 获得了 Budi Luhur 大学的学士学位和硕士学位，主修信息技术专业。

除了工作和在各种技术论坛上以及邮件上回答问题外，他会浪迹于美丽的海滩、潜水和拍水下世界照片。

“我要感谢我的父母和妻子 Nunung Astuti，感谢他们在我用个人时间来审阅本书期间对我的支持。同样也要感谢在 Budi Luhur 大学的同事，在 Kaskus 编程社区的朋友们，以及在印度尼西亚 Java User Group 的人们。愿代码与你同在 (May the Source be with you)。”

Eduardo Macarrón 作为企业集成和解决方案架构师在电力工业行业任职 15 年，专注于大型项目（超过 100 位开发人员的规模）

他是开源技术爱好者，自 2010 年起是 MyBatis 项目开发成员。

关于译者

娄变 Java 软件工程师，现居于南京。2013 年毕业于安徽师范大学计算机系，主修计算机技术专业；现从事 JavaEE 企业级应用金融方向的开发；精通 Java 和 Javascript。

作为 Java 行业小生，正在潜心修炼中。热衷于开源项目，喜欢研究和学习当前流行的开源框架。

前段时间由于工作和学习的需要，我打算深入研究 MyBatis 框架。于是在网上查找关于 MyBatis 的教程，发现国内网上关于 MyBatis 的教程资料少得可怜：除了 MyBatis 官网上的用户使用手册外，就没有比较系统地讲述 MyBatis 的教程了。

无意间发现了这本《Java Persistence with MyBatis 3》，它比较全面地讲述了 MyBatis 框架。感叹于国内 MyBatis 资料比较匮乏，故而鼓起勇气自不量力地尝试将此书翻译成中文，希望能够对国内的 MyBatis 用户有所帮助。

译者声明

本书的翻译由 娄变 完成，如对本书翻译质量有任何建议和看法，请您联系 terrylouis1991@gmail.com 或 louemail@qq.com。

本书免费发布，仅供学习和交流使用，不可用于商业用途。如果个人或者出版社有意出版此书，请先联系原作者 K. Siva Prasad Reddy 和原出版社 <http://www.packtpub.com>。原书版权归原作者所有，本书版权归译者娄变所有。

目录

前言	9
第一章 MyBatis 入门	13
1.1 MyBatis 是什么?	13
1.2 为什么选择 MyBatis?	13
1.2.1 消除大量的 JDBC 冗余代码	14
1.2.2 低学习曲线	18
1.2.3 能够很好地与传统数据库协同工作	18
1.2.4 接受 SQL	18
1.2.5 与 Spring 和 Guice 框架的集成支持	18
1.2.6 与第三方缓存类库的集成支持	18
1.2.7 良好的性能	19
1.3 MyBatis 安装和配置	19
1.3.1 新建表 STUDENTS, 插入样本数据	19
1.3.2 新建一个 Java 项目, 将 MyBatis-3.2.2.jar 添加到 classpath 中	20
1.3.3 新建 mybatis-config.xml 和映射器 StudentMapper.xml 配置文件	21
1.3.4 新建 MyBatisSqlSessionFactory 单例类	23
1.3.5 新建 StudentMapper 接口和 StudentService 类	24
1.3.6 新建一个 JUnit 测试类来测试 StudentService	26
1.3.7 它是怎么工作的	28
1.4 域模型样例	28
1.5 总结	29
第二章 引导 MyBatis	30
2.1 使用 XML 配置 MyBatis	31
2.1.1 environment	32
2.1.2 数据源 DataSource	33
2.1.3 事务管理器 TransactionManager	33
2.1.4 属性 Properties	34
2.1.5 类型别名 typeAliases	34
2.1.6 类型处理器 typeHandlers	36
2.1.7 全局参数设置 Settings	40

2.1.8 SQL 映射定义 Mappers	41
2.2 使用 Java API 配置 MyBatis	41
2.2.1 环境配置 Environment	42
2.2.2 数据源 DataSource	42
2.2.3 事务工厂 TransactionFactory	43
2.2.4 类型别名 typeAliases	44
2.2.5 类型处理器 typeHandlers	45
2.2.6 全局参数设置 Settings	45
2.2.7 Mappers	46
2.3 自定义 MyBatis 日志	46
2.4 总结	47
第三章 使用 XML 配置 SQL 映射器	48
3.1 映射器配置文件和映射器接口	49
3.2 映射语句	51
3.2.1 INSERT 语句	51
[自动生成主键]	52
3.2.2 UPDATE 语句	53
3.2.3 删除语句	54
3.2.4 SELECT 语句	55
3.3 结果集映射 ResultMaps	57
3.3.1 简单 resultMap	57
3.3.2 拓展 resultMap	59
3.4 一对一映射	60
3.4.1 使用嵌套结果 resultMap 实现一对一关系映射	62
3.4.2 使用嵌套查询实现一对一关系映射	63
3.5 一对多映射	64
3.5.1 使用内嵌结果 resultMap 实现一对多映射	65
3.5.2 使用嵌套 Select 语句实现一对多映射	66
3.6 动态 SQL	67
3.6.1 If 条件	67
3.6.2 choose,when 和 otherwise 条件	69
3.6.3 Where 条件	70

3.6.4 <trim>条件	70
3.6.5 foreach 循环	71
3.6.6 set 条件	72
3.7 MyBaits 食谱	73
3.7.1 处理枚举类型	73
3.7.2 处理 CLOB/BLOB 类型数据	74
3.7.3 传入多个输入参数	77
3.7.4 多行结果集映射成 Map	77
3.7.5 使用 RowBounds 对结果集进行分页	78
3.7.6 使用 ResultSetHandler 自定义结果集 ResultSet 处理	78
3.7.7 缓存	80
3.8 总结	81
第四章 使用注解配置 SQL 映射器	82
4.1 在映射器 Mapper 接口上使用注解	83
4.2 映射语句	83
4.2.1 @Insert	83
[自动生成主键]	83
4.2.2 @Update	84
4.2.3 @Delete	84
4.2.4 @Select	85
4.3 结果映射	85
4.3.1 一对一映射	87
4.3.2 一对多映射	88
4.4 动态 SQL	91
4.4.1 @SelectProvider	91
4.4.2 @InsertProvider	95
4.4.3 @UpdateProvider	96
4.4.4 @DeleteProvider	96
4.5 总结	97
第五章 与 Spring 集成	98
5.1 在 Spring 应用程序中配置 MyBatis	98
5.1.1 安装	98

5.1.2 配置 MyBatis Beans	100
5.2 使用 SqlSession	101
5.3 使用映射器	103
5.3.1 <mybatis:scan />	104
5.3.2 MapperScan.....	105
5.4 使用 Spring 进行事务管理	106
5.5 总结	109

前言

对很多软件系统而言，保存数据到数据库和从数据库中检索数据是其工作流程中至关重要的一部分。在 Java 领域，有很多的实现了数据持久化层的工具和框架，它们每一个都有自己不同的实现方法。而 MyBatis，一个简单但功能强大的 Java 持久化框架，则采用了消除冗余代码和充分利用 SQL 和 Java 自身提供的强大的特性的策略。

这本 MyBatis 教程将带你经历 MyBatis 的安装、配置和使用这几个过程。每一章涉及到的概念将通过简单而实用的例子配合详细的指导来解释。

在本书的最后，你不仅会学到 MyBatis 的理论知识，还会在真正的项目中使用 MyBatis 的过程中，得到动手实践的认识和体会。

这本书也可以当作参考书或者用来重新学习每一章中讨论的概念。本书还提供了一些有说明性的例子，无论它是否必要，以确保所阐述的概念容易被理解。

本书涵盖了那些内容

第一章，MyBatis 入门，介绍了 MyBatis 开源框架和解释了使用 MyBatis 而不是使用 JDBC 的优点。我们也着眼于怎样创建一个项目，在使用和不使用 Maven 构建工具的情况下安装 MyBatis 框架依赖，配置和使用 MyBatis。

第二章，引导 MyBatis，涵盖了怎样使用 XML 配置和基于 Java API 两种方式来引导 MyBatis。我们还会学到各种 MyBatis 配置项如类型别名(type alias)，类型处理器(type handlers)，全局参数设置，等等。

第三章，使用 XML 配置 SQL 映射器，本章将深入到如何使用映射器 Mapper XML 配置文件来书写 SQL 映射语句(statement)。我们将学习到怎样配置简单 SQL 语句、配置“一对一”关系和“一对多”关系的 SQL 语句、使用 ResultMaps 来映射结果。我们还会学习到怎样构造动态 SQL 语句(dynamic SQLs)，结果分页，和自定义 ResultSet 处理器。

第四章，使用注解配置 SQL 映射器，本章涵盖了怎样使用注解书写 SQL 映射语句。我们将学习到如何配置简单 SQL 语句，以及“一对一”关系和“一对多”关系的 SQL 语句。我们还会使用 SqlProvider 注解来探究构建动态 SQL。

第五章，与 Spring 的集成，本章涵盖了怎样集成 MyBatis 和 Spring 框架。我们将学习到怎样安装 Spring 类库，往 Spring 应用上下文 ApplicationContext 中注册 MyBatis beans，映射器 Mapper beans 和 SqlSession 注入，以及使用 Spring 的注解事务处理机制。

你需要为本书准备什么

运行本书上的样例，你需要安装以下软件：

- Java JDK 1.5+
- MyBatis 最新版本 (<https://code.google.com/p/mybatis/>)
- MySQL (<http://www.mysql.com/>) 或其他任意关系数据库以及相应 JDBC 驱动
- Eclipse (<http://www.eclipse.org>) 或其他你喜欢的 IDE
- Apache Maven 构建工具 (<http://maven.apache.org/>)

本书面向的读者

本书面向的人群是最起码有基本的数据库和 JDBC 使用经验的开发人员。你需要对 SQL 有基本的了解。我们不假定你先前有过使用 MyBatis 的经验。

约定

在本书中，你会发现有大量的不同样式风格的文本，以区别不同类型的信息。这里有这些样式风格的样例以及其代表的意义解释。

源码词汇将会类似如下所示：“We can include other contexts through the use of the `include` directive.” (`include` 作为源码词汇跟一般词汇样式不一样)

代码块格式设置如下：

```
Java Code
1 package com.mybatis3.domain;
2 import java.util.Date;
3 public class Student
4 {
5     private Integer studId;
6     private String name;
7     private String email;
8     private Date dob;
9     // setters and getters
10 }
11
```

当我们希望你对特定部分的代码块引起关注时，相关的行或者元素将会被加粗：

Java Code

```
1 package com.mybatis3.domain;
2 import java.util.Date;
3 public class Student
4 {
5     private Integer studId;
6     private String name;
7     private String email;
8     private Date dob;
9     // setters and getters
10 }
```

新的术语和重要词汇也会被加粗。比如，你在屏幕上，菜单上或者对话框会显示如下类似的信息：“点击 **Next** 按钮进入下一页”。



警告或者重要标记将以这样的形式展示



提示 或者小技巧 会像这样展示

读者反馈

我们一直非常欢迎读者的反馈。请告诉我们你对本书的看法-你喜欢什么和不喜欢什么。您的反馈对我们头衔的发展起到了极大的作用。

一般性的反馈，请发送邮件至 feedback@packtpub.com，请在你的邮件标题上标注下书名。

如果是一些您所擅长的话题，又或者是您有兴趣写或者赞助书籍，请在 www.packtpub.com/authors 上查看我们的作者指南。

客户支持

你现在已经是 Packt 图书引以为豪的拥有者了。我们会为您提供一系列的服务以让您的购买物有所值。

下载样例源码

你可以使用在 <http://www.packtpub.com> 的账号下载你购买过的所有 Packt 图书上的样例代码。如果你通过其渠道购买的此书，你可以访问 <http://www.packtpub.com/support>，注册一个帐号，(选择本书)，样例代码会直接通过邮箱发送给你。(译者注：读者也可以到 <http://download.csdn.net/detail/u010349169/7555959> 上下载。)

勘错

虽然我们尽全力来确保我们内容的准确性，但错误是不能避免的。如果您发现了我们任何书中的一个错误-可能是文本或者代码上的错误-您若汇报给我们，我们会非常感激！您这么做，可以其他的读者免受挫败感，帮助我们提高本书随后版本的质量。如果您发现任何勘错，请您访问 <http://www.packtpub.com/submit-errata>, 选择您购买的书籍，点击 errata submission form 超链接，提交您详细的勘错信息。一旦您的勘错被验证，您的提交将会被接受，并且勘错会被上传到我们的网站上，或者添加到该图书名下勘错列表中。任何勘错信息都可以在 <http://www.packtpub.com/support> 通过图书名查到。

盗版问题

在所有传媒手段中，互联网上的教材著作权盗版是一个正在进行的问题。在 Packt，我们非常看重对我们的著作权和许可证的保护。如果您发现任何关于我们作品的非法拷贝，不论以什么形式，或者在互联网上，请您立刻为我们提供其地址或者是网站名，以让我们寻求相应的补救措施。

(如果发现有侵权嫌疑的资料,) 请提供侵权嫌疑资料的链接, [通过 copyright@packtpub.com](mailto:copyright@packtpub.com) 与我们联系。

我们非常感谢您保护我们的作家，以及我们给你带来有价值的内容的能力的方面做出的帮助！

疑问

如果你有本书任何方面的疑问，[可以通过 questions@packtpub.com](mailto:questions@packtpub.com) 联系我们，我们会尽力处理好它。

1

第一章 MyBatis 入门

本章将涵盖以下话题：

- MyBatis 是什么？
- 为什么选择 MyBatis？
- MyBatis 安装配置
- 域模型样例

1.1 MyBatis 是什么？

MyBatis 是一个简化和实现了 Java 数据持久化层(persistence layer)的开源框架，它抽象了大量的 JDBC 冗余代码，并提供了一个简单易用的 API 和数据库交互。

MyBatis 的前身是 iBATIS，iBATIS 于 2002 年由 Clinton Begin 创建。MyBatis 3 是 iBATIS 的全新设计，支持注解和 Mapper。

MyBatis 流行的主要原因在于它的简单性和易使用性。在 Java 应用程序中，数据持久化层涉及到的工作有：将从数据库查询到的数据生成所需要的 Java 对象；将 Java 对象中的数据通过 SQL 持久化到数据库中。

MyBatis 通过抽象底层的 JDBC 代码，自动化 SQL 结果集产生 Java 对象、Java 对象的数据持久化数据库中的过程使得对 SQL 的使用变得容易。

如果你正在使用 iBATIS，并且想将 iBATIS 移植到 MyBatis 上，你可以在 MyBatis 的官方网站（<https://code.google.com/p/mybatis/wiki/DocUpgrade>）上找到详细的指导步骤。

1.2 为什么选择 MyBatis？

当前有很多 Java 实现的持久化框架，而 MyBatis 流行起来有以下原因：

- 它消除了大量的 JDBC 冗余代码
- 它有低的学习曲线
- 它能很好地与传统数据库协同工作
- 它可以接受 SQL 语句
- 它提供了与 Spring 和 Guice 框架的集成支持
- 它提供了与第三方缓存类库的集成支持
- 它引入了更好的性能

1.2.1 消除大量的 JDBC 冗余代码

Java 通过 Java 数据库连接 (Java DataBase Connectivity, JDBC) API 来操作关系型数据库, 但是 JDBC 是一个非常底层的 API, 我们需要书写大量的代码来完成对数据库的操作。

让我们演示一下我们是怎样使用纯的 JDBC 来对表 STUDENTS 实现简单的 select 和 insert 操作的。

假设表 STUDENTS 有 STUD_ID, NAME, EMAIL 和 DOB 字段。对应的 Student JavaBean 定义如下:

Java Code

```
1 package com.mybatis3.domain;
2 import java.util.Date;
3 public class Student
4 {
5     private Integer studId;
6     private String name;
7     private String email;
8     private Date dob;
9     // setters and getters
10 }
11
```

下载用例源码



你可以通过你所购买的所有的 Packt 书籍上自带的账户, 到网站 <http://www.packtpub.com> 上下载书籍中的用例源码文件。如果你是通过其他途径购买的此书, 可以访问 <http://www.packtpub.com/support>, 注册一个账户, (选择该书), 代码会直接发送到你的邮箱。(读者也可以到 <http://download.csdn.net/detail/u010349169/7555959> 上下载)

下面的 StudentService.java 实现了 通过 JDBC 对表 STUDENTS 的 SELECT 和 INSERT 操作:

Java Code

```
1 public Student findStudentById(int studId)
2 {
3     Student student = null;
4     Connection conn = null;
5     try
6     {
7         //获得数据库连接
8         conn = getDatabaseConnection();
```

```

9      String sql = "SELECT * FROM STUDENTS WHERE STUD_ID=?";
10     //创建 PreparedStatement
11     PreparedStatement pstmt = conn.prepareStatement(sql);
12     //设置输入参数
13     pstmt.setInt(1, studId);
14     ResultSet rs = pstmt.executeQuery();
15     //从数据库中取出结果并生成 Java 对象
16     if(rs.next())
17     {
18         student = new Student();
19         student.setStudId(rs.getInt("stud_id"));
20         student.setName(rs.getString("name"));
21         student.setEmail(rs.getString("email"));
22         student.setDob(rs.getDate("dob"));
23     }
24 }
25 catch (SQLException e)
26 {
27     throw new RuntimeException(e);
28 }
29 finally
30 {
31     //关闭连接
32     if(conn != null)
33     {
34         try
35         {
36             conn.close();
37         }
38         catch (SQLException e) { }
39     }
40 }
41 return student;
42 }
43
44 public void createStudent(Student student)
45 {
46     Connection conn = null;
47     try
48     {
49         //获得数据库连接
50         conn = getDatabaseConnection();
51         String sql = "INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL,DOB)
52             VALUES(?,?,?,?)";

```

```

53      //创建 PreparedStatement
54      PreparedStatement pstmt = conn.prepareStatement(sql);
55      //设置输入参数
56      pstmt.setInt(1, student.getStudId());
57      pstmt.setString(2, student.getName());
58      pstmt.setString(3, student.getEmail());
59      pstmt.setDate(4, new
60          java.sql.Date(student.getDob().getTime()));
61      pstmt.executeUpdate();
62  }
63  catch (SQLException e)
64  {
65      throw new RuntimeException(e);
66  }
67  finally
68  {
69      //关闭连接
70      if(conn != null)
71      {
72          try
73          {
74              conn.close();
75          }
76          catch (SQLException e) { }
77      }
78  }
79  }
80  protected Connection getDatabaseConnection() throws SQLException
81  {
82      try
83      {
84          Class.forName("com.mysql.jdbc.Driver");
85          return DriverManager.getConnection
86              ("jdbc:mysql://localhost:3306/test", "root", "admin");
87      }
88      catch (SQLException e)
89      {
90          throw e;
91      }
92      catch (Exception e)
93      {
94          throw new RuntimeException(e);
95      }
96  }

```


上述的每个方法中有大量的重复代码：**创建一个连接，创建一个 Statement 对象，设置输入参数，关闭资源**（如 connection, statement, resultSet）。

MyBatis 抽象了上述的这些相同的任务，如准备需要被执行的 SQL statement 对象并且将 Java 对象作为输入数据传递给 statement 对象的任务，进而开发人员可以专注于真正重要的方面。

另外，MyBatis 自动化了将从输入的 Java 对象中的属性设置成查询参数、从 SQL 结果集上生成 Java 对象这两个过程。

现在让我们看看怎样通过 MyBatis 实现上述的方法：

1. 在 SQL Mapper 映射配置文件中配置 SQL 语句，假定为 StudentMapper.xml

XML Code

```

1 <select id="findStudentById" parameterType="int" resultType="Student">
2     SELECT STUD_ID AS studId, NAME, EMAIL, DOB
3     FROM STUDENTS WHERE STUD_ID=#{Id}
4 </select>
5 <insert id="insertStudent" parameterType="Student">
6     INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL,DOB)
7     VALUES("#{studId},#{name},#{email},#{dob})
8 </insert>
9

```

2. 创建一个 StudentMapper 接口。

Java Code

```

1 public interface StudentMapper
2 {
3     Student findStudentById(Integer id);
4     void insertStudent(Student student);
5 }

```

3. 在 Java 代码中，你可以使用如下代码触发 SQL 语句：

Java Code

```

1 SqlSession session = getSqlSessionFactory().openSession();
2 StudentMapper mapper = session.getMapper(StudentMapper.class);
3 // Select Student by Id
4 Student student = mapper.selectStudentById(1);
5 //To insert a Student record
6 mapper.insertStudent(student);

```

就是这么简单！你不需要创建 Connection 连接，PreparedStatement，不需要自己对每一次数据库操作进行手动设置参数和关闭连接。只需要配置数据库连接属性和 SQL 语句，MyBatis 会处理这些底层工作。

现在你不用担心 SqlSessionFactory、SqlSession、Mapper XML 文件是什么，这些概念我们会在接下来的章节中详细解释。

另外，MyBatis 还提供了其他的一些特性来简化持久化逻辑的实现：

- 它支持复杂的 SQL 结果集数据映射到嵌套对象图结构
- 它支持一对一和一对多的结果集和 Java 对象的映射
- 它支持根据输入的数据构建动态的 SQL 语句

1.2.2 低学习曲线

MyBatis 能够流行的首要原因之一在于它学习和使用起来非常简单，它取决于你 Java 和 SQL 方面的知识。如果开发人员很熟悉 Java 和 SQL，他们会发现 MyBatis 入门非常简单。

1.2.3 能够很好地与传统数据库协同工作

有时我们可能需要用不正规形式与传统数据库协同工作，使用成熟的 ORM 框架（如 Hibernate）有可能、但是很难跟传统数据库很好地协同工作，因为他们尝试将 Java 对象静态地映射到数据库的表上¹。

而 MyBatis 是将查询的结果与 Java 对象映射起来，这使得 MyBatis 可以很好地与传统数据库协同工作。你可以根据面相对象的模型创建 Java 域对象，执行传统数据库的查询，然后将结果映射到对应的 Java 对象上。

1.2.4 接受 SQL

成熟的 ORM 框架（如 Hibernate）鼓励使用实体对象（Entity Objects）和在其底层自动产生 SQL 语句。由于这种的 SQL 生成方式，我们有可能不能够利用到数据库的一些特有的特性。Hibernate 允许执行本地 SQL，但是这样会打破持久层和数据库独立的原则。

MyBatis 框架接受 SQL 语句，而不是将其对开发人员隐藏起来。由于 MyBatis 不会产生任何的 SQL 语句，所以开发人员就要准备 SQL 语句，这样就可以充分利用数据库特有的特性并且可以准备自定义的查询。另外，MyBatis 对存储过程也提供了支持。

1.2.5 与 Spring 和 Guice 框架的集成支持

MyBatis 提供了与流行的依赖注入框架 Spring 和 Guice 的开包即用的集成支持，这将进一步简化 MyBatis 的使用

1.2.6 与第三方缓存类库的集成支持

MyBatis 有内建的 SqlSession 级别的缓存机制，用于缓存 Select 语句查询出来的结果。除此之外，MyBatis 提供了与多种第三方缓存类库的集成支持，如 EHCache，OSCache，Hazelcast。

¹ 这一类型的框架都尝试将一张表的一条记录映射成一个 JavaBean，这种方式一定程度上丧失了 SQL 语句的灵活性

1.2.7 良好的性能

性能问题是关乎软件应用成功与否的关键因素之一。为了达到更好的性能，需要考虑很多事情，而对很多应用而言，数据持久化层是整个系统性能的关键。

MyBatis 支持数据库连接池，消除了为每一个请求创建一个数据库连接的开销

MyBatis 提供了内建的缓存机制，在 `SqlSession` 级别提供了对 SQL 查询结果的缓存。即：如果你调用了相同的 `select` 查询，MyBatis 会将放在缓存的结果返回，而不会再去查询数据库。

MyBatis 框架并没有大量地使用代理机制²，因此对于其他的过度地使用代理的 ORM 框架而言，MyBatis 可以获得更好的性能。



在软件开发中，并没有通用的（一体适用）的解决方案，每一个应用会有不同的一系列的要求，而我们应该根据应用的需要来选择我们的工具和框架。在前一节中，我们看到了使用 MyBatis 的优点。然而也有一些情况，MyBatis 并不是理想的或者是最好的解决方案。

如果你的应用是以面向对象模型，并且向动态生成 SQL 语句，那么 MyBatis 可能就不符合你的要求。另外，如果你想让你的应用有一个传递性的缓存机制的话（保存父对象时也应该保存关联的子对象），Hibernate 会更适合你。

1.3 MyBatis 安装和配置

我们假设你的系统上已经安装了 JDK1.6+ 和 MySQL5。JDK 和 MySQL 安装过程不在本书的叙述范围。

在写本书时，MyBatis 最新版是 MyBatis 3.2.2。贯穿本书，我们将使用 MyBatis 3.2.2 版本。

本文并不限定你使用什么类型的 IDE（如 Eclipse，NetBeans IDE，或者 IntelliJ IDEA，它们通过提供自动完成，重构，调试特性来很大程度上简化了开发）来编码，你可以选择你喜欢的 IDE。

本节将（通过以下步骤）说明如何使用 MyBatis 开发一个简单的 Java 项目：

- 新建表 STUDENTS，插入样本数据
- 新建一个 Java 项目，将 MyBatis-3.2.2.jar 添加到 classpath 中
- 新建建 MyBatisSqlSessionFactory 单例模式类
- 新建映射器 StudentMapper 接口和 StudentService 类
- 新建一个 JUnit 测试类来测试 StudentService

1.3.1 新建表 STUDENTS，插入样本数据

使用以下 SQL 脚本往 MySQL 数据库中创建 STUDENTS 表插入样本数据：

SQL Code

² 一些 ORM 框架使用了大量的动态代理模式来产生实体对象等，由于动态代理本身有很大的内存消耗，大量使用动态代理，会使整个系统性能变得很差。

```

1 CREATE TABLE STUDENTS
2 (
3   stud_id int(11) NOT NULL AUTO_INCREMENT,
4   name varchar(50) NOT NULL,
5   email varchar(50) NOT NULL,
6   dob date DEFAULT NULL,
7   PRIMARY KEY (stud_id)
8 ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
9 /*Sample Data for the students table */
10 insert into students(stud_id,name,email,dob)
11 values (1,'Student1','student1@gmail.com','1983-06-25');
12 insert into students(stud_id,name,email,dob)
13 values (2,'Student2','student2@gmail.com','1983-06-25');

```

1.3.2 新建一个 Java 项目，将 MyBatis-3.2.2.jar 添加到 classpath 中

让我们创建一个 Java 项目，并被指 MyBatis JAR 包依赖：

1. 创建一个 Java 项目，名为 mybatis-demo。
2. 如果你没有使用类似于 Maven 和 Gradle 之类的依赖管理构建工具，你需要手动下载这些依赖的 JAR 包，手动添加到 classpath 中。
3. 你可以从 <http://code.google.com/p/mybatis> 上下载 MyBatis 的发布包 mybatis-3.2.2.zip。这个包包含了 mybatis-3.2.2.jar 文件和它的可选的依赖包如 slf4j/log4j 日志 jar 包。
4. 我们将使用 SLF4J 日志记录框架 和 log4j 一起记录日志。mybatis-3.2.2.zip 包含了 slf4j 这个依赖 jar 包。
5. 解压 mybatis-3.2.2.zip 文件，将 mybatis-3.2.2.jar, lib/slf4j-api-1.7.5.jar, lib/slf-log4j12-1.7.5.jar, 和 lib/log4j-1.2.17.jar 这些 jar 包添加到 classpath 中
6. 你可以从 <http://junit.org> 上下载 JUnit JAR 文件，从 <http://www.mysql.com/downloads/connector/j/> 上下载 MySQL 数据库驱动
7. 将 junit-4.11.jar 和 mysql-connector-java-5.1.22.jar 添加到 classpath 中
8. 如果你正在使用 maven，配置这些 jar 包依赖就变得简单多了。在你的 pom.xml 中添加以下依赖即可：

XML Code

```

1 <dependencies>
2   <dependency>
3     <groupId>org.mybatis</groupId>
4     <artifactId>mybatis</artifactId>
5     <version>3.2.2</version>
6   </dependency>
7   <dependency>
8     <groupId>mysql</groupId>
9     <artifactId>mysql-connector-java</artifactId>
10    <version>5.1.22</version>
11    <scope>runtime</scope>
12  </dependency>

```

```

13 <dependency>
14   <groupId>org.slf4j</groupId>
15   <artifactId>slf4j-api</artifactId>
16   <version>1.7.5</version>
17 </dependency>
18 <dependency>
19   <groupId>org.slf4j</groupId>
20   <artifactId>slf4j-log4j12</artifactId>
21   <version>1.7.5</version>
22   <scope>runtime</scope>
23 </dependency>
24 <dependency>
25   <groupId>log4j</groupId>
26   <artifactId>log4j</artifactId>
27   <version>1.2.17</version>
28   <scope>runtime</scope>
29 </dependency>
30 <dependency>
31   <groupId>junit</groupId>
32   <artifactId>junit</artifactId>
33   <version>4.11</version>
34   <scope>test</scope>
35 </dependency>
36 </dependencies>
37

```

9. 新建 log4j.properties 文件，添加到 classpath 中。

NormalText Code

```

1 log4j.rootLogger=DEBUG, stdout
2 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
3 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
4 log4j.appender.stdout.layout.ConversionPattern=%d [%-5p] %c - %m%n

```

1.3.3 新建 mybatis-config.xml 和映射器 StudentMapper.xml 配置文件

让我们来创建 MyBatis 的主要配置文件 mybatis-config.xml，其中包括数据库连接信息，类型别名等等；然后创

建一个包含了映射的 SQL 语句的 StudentMapper.xml 文件。

1. 创建 MyBatis 的主要配置文件 mybatis-config.xml，其中包括数据库连接信息，类型别名等等，然后将其加到 classpath 中；

XML Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
3 "http://mybatis.org/dtd/mybatis-3-config.dtd">
4 <configuration>
5   <typeAliases>
6     <typeAlias alias="Student" type="com.mybatis3.domain.Student" />
7   </typeAliases>
8   <environments default="development">
9     <environment id="development">
10      <transactionManager type="JDBC" />
11      <dataSource type="POOLED">
12        <property name="driver" value="com.mysql.jdbc.Driver" />
13        <property name="url" value="jdbc:mysql://localhost:3306/test" />
14        <property name="username" value="root" />
15        <property name="password" value="admin" />
16      </dataSource>
17    </environment>
18  </environments>
19  <mappers>
20    <mapper resource="com/mybatis3/mappers/StudentMapper.xml" />
21  </mappers>
22 </configuration>
23
```

2. 创建 SQL 映射器 XML 配置文件 StudentMapper.xml 并且将它放在 com.mybatis3.mappers 包中

XML Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4 <mapper namespace="com.mybatis3.mappers.StudentMapper">
5   <resultMap type="Student" id="StudentResult">
6     <id property="studId" column="stud_id" />
7     <result property="name" column="name" />
8     <result property="email" column="email" />
9     <result property="dob" column="dob" />
10  </resultMap>
11  <select id="findAllStudents" resultMap="StudentResult">
```

```

12     SELECT * FROM STUDENTS
13 </select>
14 <select id="findStudentById" parameterType="int" resultType="Student">
15     SELECT STUD_ID AS STUDID, NAME, EMAIL, DOB
16     FROM STUDENTS WHERE STUD_ID=#{Id}
17 </select>
18 <insert id="insertStudent" parameterType="Student">
19     INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL,DOB)
20     VALUES("#{studId } ,#{name},#{email},#{dob})
21 </insert>
22 </mapper>
23

```

上述的 StudentMapper.xml 文件包含的映射的 SQL 语句可以通过 ID 加上名空间调用。

1.3.4 新建 MyBatisSqlSessionFactory 单例类

新建 MyBatisSqlSessionFactory.java 类文件，实例化它，使其持有一个 SqlSessionFactory 单例对象：

Java Code

```

1 package com.mybatis3.util;
2 import java.io.*;
3 import org.apache.ibatis.io.Resources;
4 import org.apache.ibatis.session.*;
5 public class MyBatisSqlSessionFactory
6 {
7     private static SqlSessionFactory sqlSessionFactory;
8     public static SqlSessionFactory getSqlSessionFactory()
9     {
10         if(sqlSessionFactory == null)
11         {
12             InputStream inputStream;
13             try
14             {
15                 inputStream = Resources.
16                     getResourceAsStream("mybatis-config.xml");
17                 sqlSessionFactory = new
18                     SqlSessionFactoryBuilder().build(inputStream);
19             }
20             catch (IOException e)
21             {
22                 throw new RuntimeException(e.getCause());
23             }
24         }
25     }
26 }

```

```

22     }
23     }
24     return sqlSessionFactory;
25 }
26 public static SqlSession openSession()
27 {
28     return getSqlSessionFactory().openSession();
29 }
30 }
31

```

上述的代码段中，我们创建了一个 `SqlSessionFactory` 对象，我们将使用它来获得 `SqlSession` 对象和执行映射的 SQL 语句。

1.3.5 新建 StudentMapper 接口和 StudentService 类

让我们创建一个 `StudentMapper` 接口，其定义的方法名和在 `Mapper XML` 配置文件定义的 SQL 映射语句名称相同；在创建一个 `StudentService.java` 类，包含了一些业务操作的实现。

1. 首先，创建 `JavaBean Student.java`

Java Code

```

1 package com.mybatis3.domain;
2 import java.util.Date;
3 public class Student
4 {
5     private Integer studId;
6     private String name;
7     private String email;
8     private Date dob;
9     // setters and getters
10 }

```

2. 创建映射器 `Mapper` 接口 `StudentMapper.java` 其方法签名和 `StudentMapper.xml` 中定义的 SQL 映射定义名相同

Java Code

```

1 package com.mybatis3.mappers;
2 import java.util.List;
3 import com.mybatis3.domain.Student;
4 public interface StudentMapper

```



```

5 {
6     List<Student> findAllStudents();
7     Student findStudentById(Integer id);
8     void insertStudent(Student student);
9 }

```

3. 现在创建 StudentService.java 实现对表 STUDENTS 的数据库操作

Java Code

```

1 package com.mybatis3.services;
2 import java.util.List;
3 import org.apache.ibatis.session.SqlSession;
4 import org.slf4j.Logger;
5 import org.slf4j.LoggerFactory;
6 import com.mybatis3.domain.Student;
7 import com.mybatis3.mappers.StudentMapper;
8 import com.mybatis3.util.MyBatisSqlSessionFactory;
9 public class StudentService
10 {
11     private Logger logger =
12         LoggerFactory.getLogger(getClass());
13     public List<Student> findAllStudents()
14     {
15         SqlSession sqlSession =
16             MyBatisSqlSessionFactory.openSession();
17         try
18         {
19             StudentMapper studentMapper =
20                 sqlSession.getMapper(StudentMapper.class);
21             return studentMapper.findAllStudents();
22         }
23         finally
24         {
25             //If sqlSession is not closed
26             //then database Connection associated this sqlSession will not be
27             returned to pool
28             //and application may run out of connections.
29             sqlSession.close();
30         }
31     }
32     public Student findStudentById(Integer studId)
33     {
34         logger.debug("Select Student By ID :{}", studId);

```

```

35     SqlSession sqlSession =
36         MyBatisSqlSessionFactory.openSession();
37     try
38     {
39         StudentMapper studentMapper =
40             sqlSession.getMapper(StudentMapper.class);
41         return studentMapper.findStudentById(studId);
42     }
43     finally
44     {
45         sqlSession.close();
46     }
47 }
48 public void createStudent(Student student)
49 {
50     SqlSession sqlSession =
51         MyBatisSqlSessionFactory.openSession();
52     try
53     {
54         StudentMapper studentMapper =
55             sqlSession.getMapper(StudentMapper.class);
56         studentMapper.insertStudent(student);
57         sqlSession.commit();
58     }
59     finally
60     {
61         sqlSession.close();
62     }
63 }
64 }

```

你也可以通过不通过 Mapper 接口执行映射的 SQL 语句。

Java Code

```

1 Student student = (Student)sqlSession.
2     selectOne("com.mybatis3.mappers.StudentMapper.findStudentById",
3         studId);

```

然而，使用 Mapper 接口是最佳实践，我们可以以类型安全的方式调用映射的 SQL 语句。

1.3.6 新建一个 JUnit 测试类来测试 StudentService

新建一个 JUnit 测试类测试 StudentService.java 中定义的方法。

Java Code

```
1 package com.mybatis3.services;
2 import java.util.*;
3 import org.junit.*;
4 import com.mybatis3.domain.Student;
5 public class StudentServiceTest
6 {
7     private static StudentService studentService;
8     @BeforeClass
9     public static void setup()
10    {
11        studentService = new StudentService();
12    }
13    @AfterClass
14    public static void teardown()
15    {
16        studentService = null;
17    }
18    @Test
19    public void testFindAllStudents()
20    {
21        List<Student> students = studentService.findAllStudents();
22        Assert.assertNotNull(students);
23        for (Student student : students)
24        {
25            System.out.println(student);
26        }
27    }
28    @Test
29    public void testFindStudentById()
30    {
31        Student student = studentService.findStudentById(1);
32        Assert.assertNotNull(student);
33        System.out.println(student);
34    }
35    @Test
36    public void testCreateStudent()
37    {
38        Student student = new Student();
39        int id = 3;
40        student.setStudId(id);
41        student.setName("student_" + id);
42        student.setEmail("student_" + id + "gmail.com");
```

```
43     student.setDob(new Date());
44     studentService.createStudent(student);
45     Student newStudent = studentService.findStudentById(id);
46     Assert.assertNotNull(newStudent);
47 }
48 }
```

1.3.7 它是怎么工作的

首先，我们配置了 MyBatis 最主要的配置文件-mybatis-config.xml,里面包含了 JDBC 连接参数；配置了映射器 Mapper XML 配置文件文件，里面包含了 SQL 语句的映射。

我们使用 mybatis-config.xml 内的信息创建了 SqlSessionFactory 对象。每个数据库环境应该就一个 SqlSessionFactory 对象实例，所以我们使用了单例模式只创建一个 SqlSessionFactory 实例。

我们创建了一个映射器 Mapper 接口-StudentMapper，其定义的方法签名和在 StudentMapper.xml 中定义的完全一样（即映射器 Mapper 接口中的方法名跟 StudentMapper.xml 中的 id 的值相同）。注意 StudentMapper.xml 中 namespace 的值被设置成 com.mybatis3.mappers.StudentMapper，是 StudentMapper 接口的完全限定名。这使我们可以使用接口来调用映射的 SQL 语句。

在 StudentService.java 中，我们在每一个方法中创建了一个新的 SqlSession，并在方法功能完成后关闭 SqlSession。每一个线程应该有它自己的 SqlSession 实例。SqlSession 对象实例不是线程安全的，并且不被共享。所以 SqlSession 的作用域最好就是其所在方法的作用域。从 Web 应用程序角度上看，SqlSession 应该存在于 request 级别作用域上。

1.4 域模型样例

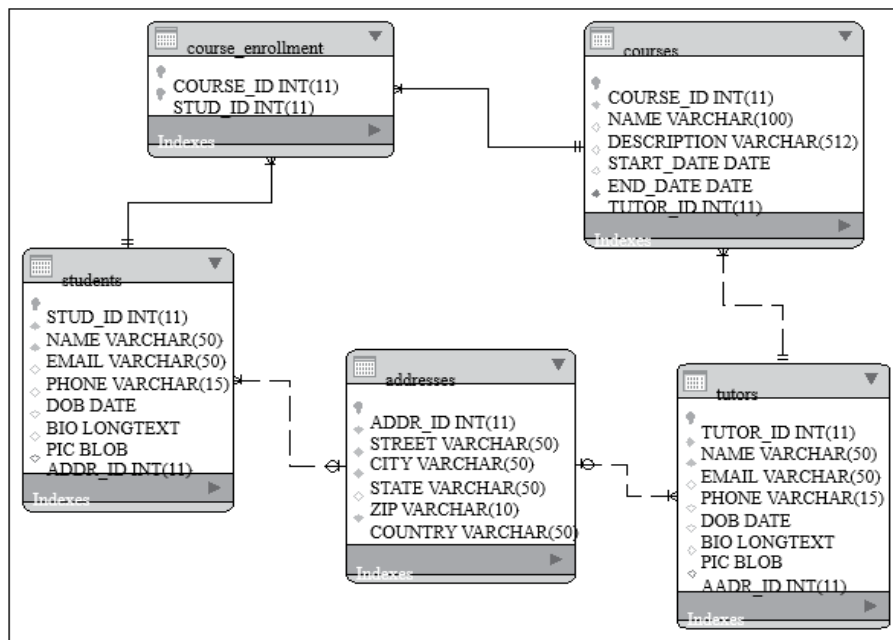
在这一节，我们将讨论一个表示**在线学习系统应用的域模型**样例，该模型的使用会贯穿全书。

在线学习系统中，学生可以选择课程，并且通过基于 Web 的传播媒介上课，是虚拟课堂或者是桌面共享系统。

有兴趣通过在线学习系统上课的导师可以在系统中注册，声明他们要教的课程明细。

课程明细包括课程名称，课程描述，课程时长。全球的学生都可以注册和选择他们想学的课程。

下面的图表表示了我们的在线学习系统的数据库结构：



1.5 总结

在本章中，我们讨论了 MyBatis，和与使用纯 JDBC 相比，使用 MyBatis 访问数据库的优点。我们学习了怎样创建一个项目，安装 MyBatis jar 包依赖，创建 MyBatis 配置文件，以及在映射器 Mapper XML 文件中配置 SQL 映射语句。我们创建了一个使用 MyBatis 对数据库插数据和取数据的 Service 类。我们创建了一个 JUnit 测试类来测试 Service 类。

2

第二章 引导 MyBatis

MyBatis 最关键的组成部分是 `SqlSessionFactory`，我们可以从中获取 `SqlSession`，并执行映射的 SQL 语句。`SqlSessionFactory` 对象可以通过基于 XML 的配置信息或者 Java API 创建。

我们将探索各种 MaBatis 配置元素，如 `dataSource`，`environments`，全局参数设置，`typeAlias`，`typeHandlers`，SQL 映射；接着我们将实例化 `SqlSessionFactory`。

本章将涵盖一下内容：

- 使用 XML 配置 MyBatis
- 使用 Java API 配置 MyBatis
- 自定义 MyBatis 日志

2.1 使用 XML 配置 MyBatis

构建 `SqlSessionFactory` 最常见的方式是基于 XML 配置 (的构造方式)。下面的 `mybatis-config.xml` 展示了一个典型的 MyBatis 配置文件的样子：

XML Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
3 "http://mybatis.org/dtd/mybatis-3-config.dtd">
4 <configuration>
5   <properties resource="application.properties">
6     <property name="username" value="db_user" />
7     <property name="password" value="verysecurepwd" />
8   </properties>
9   <settings>
10    <setting name="cacheEnabled" value="true" />
11  </settings>
12  <typeAliases>
13    <typeAlias alias="Tutor" type="com.mybatis3.domain.Tutor" />
14    <package name="com.mybatis3.domain" />
15  </typeAliases>
16  <typeHandlers>
17    <typeHandler handler="com.mybatis3.typehandlers. PhoneTypeHandler" />
18    <package name="com.mybatis3.typehandlers" />
19  </typeHandlers>
20  <environments default="development">
21    <environment id="development">
22      <transactionManager type="JDBC" />
23      <dataSource type="POOLED">
24        <property name="driver" value="${jdbc.driverClassName}" />
25        <property name="url" value="${jdbc.url}" />
26        <property name="username" value="${jdbc.username}" />
27        <property name="password" value="${jdbc.password}" />
28      </dataSource>
29    </environment>
30    <environment id="production">
31      <transactionManager type="MANAGED" />
32      <dataSource type="JNDI">
33        <property name="data_source" value="java:comp/jdbc/MyBatisDemoDS" />
34      </dataSource>
35    </environment>
36  </environments>
37  <mappers>
38    <mapper resource="com/mybatis3/mappers/StudentMapper.xml" />
```

```

39     <mapper url="file:///D:/mybatisdemo/mappers/TutorMapper.xml" />
40     <mapper class="com.mybatis3.mappers.TutorMapper" />
41 </mappers>
42 </configuration>
43

```

下面让我们逐个讨论上述配置文件的组成部分，先从最重要的部分开始，即 **environments**：

2.1.1 environment

MyBatis 支持配置多个 `dataSource` 环境，可以将应用部署到不同的环境上，如 DEV(开发环境)，TEST(测试环境)，QA(质量评估环境)，UAT(用户验收环境)，PRODUCTION(生产环境)，可以通过将默认 `environment` 值设置成想要的 `environment id` 值。

在上述的配置中，默认的环境 `environment` 被设置成 `development`。当需要将程序部署到生产服务器上时，你不需要修改什么配置，只需要将默认环境 `environment` 值设置成生产环境的 `environment id` 属性即可。

有时候，我们可能需要在相同的应用下使用多个数据库。比如我们可能有 `SHOPPING-CART` 数据库来存储所有的订单明细；使用 `REPORTS` 数据库存储订单明细的合计，用作报告。

如果你的应用需要连接多个数据库，你需要将每个数据库配置成独立的环境，并且为每一个数据库创建一个 `SqlSessionFactory`。

XML Code

```

1 <environments default="shoppingcart">
2   <environment id="shoppingcart">
3     <transactionManager type="MANAGED" />
4     <dataSource type="JNDI">
5       <property name="data_source" value="java:comp/jdbc/ ShoppingcartDS" />
6     </dataSource>
7   </environment>
8   <environment id="reports">
9     <transactionManager type="MANAGED" />
10    <dataSource type="JNDI">
11      <property name="data_source" value="java:comp/jdbc/ReportsDS" />
12    </dataSource>
13  </environment>
14 </environments>
15

```

我们可以如下为每个环境创建一个 `SqlSessionFactory`：

Java Code

```

1 inputStream = Resources.getResourceAsStream("mybatis-config.xml");
2 defaultSqlSessionFactory = new SqlSessionFactoryBuilder().
3   build(inputStream);
4 cartSqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStre
5     am, "shoppingcart");

```



```
6 reportSqlSessionFactory = new SqlSessionFactoryBuilder().
7 build(inputStream, "reports");
```

创建 `SqlSessionFactory` 时，如果没有明确指定环境 `environment id`，则会使用默认的环境 `environment` 来创建。在上述的源码中，默认的 `SqlSessionFactory` 便是使用 `shoppingcart` 环境设置创建的。

对于每个环境 `environment`，我们需要配置 `dataSource` 和 `transactionManager` 元素。

2.1.2 数据源 DataSource

`dataSource` 元素被用来配置数据库连接属性。

XML Code

```
1 <dataSource type="POOLED">
2   <property name="driver" value="${jdbc.driverClassName}" />
3   <property name="url" value="${jdbc.url}" />
4   <property name="username" value="${jdbc.username}" />
5   <property name="password" value="${jdbc.password}" />
6 </dataSource>
7
```

`dataSource` 的类型可以配置成其内置类型之一，如 `UNPOOLED`，`POOLED`，`JNDI`。

- 如果将类型设置成 `UNPOOLED`，MyBatis 会为每一个数据库操作创建一个新的连接，并关闭它。该方式适用于只有小规模数量并发用户的简单应用程序上。
- 如果将属性设置成 `POOLED`，MyBatis 会创建一个数据库连接池，连接池中的一个连接将会被用作数据库操作。一旦数据库操作完成，MyBatis 会将此连接返回给连接池。在开发或测试环境中，经常使用此种方式。
- 如果将类型设置成 `JNDI`，MyBatis 从在应用服务器向配置好的 `JNDI` 数据源 `dataSource` 获取数据库连接。在生产环境中，优先考虑这种方式。

2.1.3 事务管理器 TransactionManager

MyBatis 支持两种类型的事务管理器：`JDBC` and `MANAGED`。

- `JDBC` 事务管理器被用作当应用程序负责管理数据库连接的生命周期（提交、回退等等）的时候。当你将 `TransactionManager` 属性设置成 `JDBC`，MyBatis 内部将使用 `JdbcTransactionFactory` 类创建 `TransactionManager`。例如，部署到 Apache Tomcat 的应用程序，需要应用程序自己管理事务。
- `MANAGED` 事务管理器是当由应用服务器负责管理数据库连接生命周期的时候使用。当你将 `TransactionManager` 属性设置成 `MANAGED` 时，MyBatis 内部使用 `ManagedTransactionFactory` 类创建事务管理器 `TransactionManager`。例如，当一个 JavaEE 的应用程序部署在类似 JBoss、WebLogic、GlassFish 应用服务器上时，它们会使用 EJB 进行应用服务器的事务管理能力。在这些管理环境中，你可以使用 `MANAGED` 事务管理器。
(译者注：Managed 是托管的意思，即是应用本身不去管理事务，而是把事务管理交给应用所在的服务器进行管理。)

2.1.4 属性 Properties

属性配置元素可以将配置值具体化到一个属性文件中，并且使用属性文件的 key 名作为占位符。在上述的配置中，我们将数据库连接属性具体化到了 application.properties 文件中，并且为 driver，URL 等属性使用了占位符。

1. 在 application.properties 文件中配置数据库连接参数，如下所示：

NormalText Code

```
1 jdbc.driverClassName=com.mysql.jdbc.Driver
2 jdbc.url=jdbc:mysql://localhost:3306/mybatisdemo
3 jdbc.username=root
4 jdbc.password=admin
```

2. 在 mybatis-config.xml 文件中，为属性使用 application.properties 文件中定义的占位符：

XML Code

```
1 <properties resource="application.properties">
2   <property name="jdbc.username" value="db_user" />
3   <property name="jdbc.password" value="verysecurepwd" />
4 </properties>
5 <dataSource type="POOLED">
6   <property name="driver" value="${jdbc.driverClassName}" />
7   <property name="url" value="${jdbc.url}" />
8   <property name="username" value="${jdbc.username}" />
9   <property name="password" value="${jdbc.password}" />
10 </dataSource>
11
```

并且，你可以在<properties>元素中配置默认参数的值。如果<properties>中定义的元素和属性文件定义元素的 key 值相同，它们会被属性文件中定义的值覆盖。

XML Code

```
1 <properties resource="application.properties">
2   <property name="jdbc.username" value="db_user" />
3   <property name="jdbc.password" value="verysecurepwd" />
4 </properties>
5
```

这里，如果 application.properties 文件包含值 jdbc.username 和 jdbc.password，则上述定义的 username 和 password 的值 db_user 和 verysecurepwd 将会被 application.properties 中定义的对应的 jdbc.username 和 jdbc.password 值覆盖。

2.1.5 类型别名 typeAliases

在 SQLMapper 配置文件中，对于 resultType 和 parameterType 属性值，我们需要使用 JavaBean 的完全限定名。

如下例子所示：

XML Code

```
1 <select id="findStudentById" parameterType="int"
2     resultType="com.mybatis3.domain.Student">
3     SELECT STUD_ID AS ID, NAME, EMAIL, DOB
4     FROM STUDENTS WHERE STUD_ID=#{Id}
5 </select>
6 <update id="updateStudent" parameterType="com.mybatis3.domain. Student">
7     UPDATE STUDENTS
8     SET NAME=#{name}, EMAIL=#{email}, DOB=#{dob}
9     WHERE STUD_ID=#{id}
10 </update>
11
```

这里我们为 resultType 和 parameterType 属性值设置为 Student 类型的完全限定名：

com.mybatis3.domain.Student

我们可以为完全限定名取一个别名 (alias)，然后其需要使用完全限定名的地方使用别名，而不是到处使用完全限定名。如下例子所示，为完全限定名起一个别名：

XML Code

```
1 <typeAliases>
2   <typeAlias alias="Student" type="com.mybatis3.domain.Student" />
3   <typeAlias alias="Tutor" type="com.mybatis3.domain.Tutor" />
4   <package name="com.mybatis3.domain" />
5 </typeAliases>
6
```

然后在 SQL Mapper 映射文件中，如下使用 Student 的别名：

XML Code

```
1 <select id="findStudentById" parameterType="int" resultType="Student">
2     SELECT STUD_ID AS ID, NAME, EMAIL, DOB
3     FROM STUDENTS WHERE STUD_ID=#{id}
4 </select>
5 <update id="updateStudent" parameterType="Student">
6     UPDATE STUDENTS
7     SET NAME=#{name}, EMAIL=#{email}, DOB=#{dob}
8     WHERE STUD_ID=#{id}
9 </update>
10
```

你可以不用为每一个 JavaBean 单独定义别名，你可以为提供需要取别名的 JavaBean 所在的包(package)，MyBatis 会自动扫描包内定义的 JavaBeans，然后分别为 JavaBean 注册一个小写字母开头的非完全限定的类名形式的别名。如下所示，提供一个需要为 JavaBeans 起别名的包名：

XML Code

```
1 <typeAliases>
2   <package name="com.mybatis3.domain" />
3 </typeAliases>
4
```

如果 Student.java 和 Tutor.java Bean 定义在 com.mybatis3.domain 包中，则 com.mybatis3.domain.Student 的别名会被注册为 student。而 com.mybatis3.domain.Tutor 别名将会被注册为 tutor。示例如下：

XML Code

```
1 <typeAliases>
2   <typeAlias alias="Student" type="com.mybatis3.domain.Student" />
3   <typeAlias alias="Tutor" type="com.mybatis3.domain.Tutor" />
4   <package name="com.mybatis3.domain" />
5   <package name="com.mybatis3.webservices.domain" />
6 </typeAliases>
7
```

还有另外一种方式为 JavaBeans 起别名，使用注解@Alias：

Java Code

```
1 @Alias("StudentAlias")
2 public class Student
3 {
4 }
```

@Alias 注解将会覆盖配置文件中的<typeAliases>定义。

2.1.6 类型处理器 typeHandlers

如上一章已经讨论过，MyBatis 通过抽象 JDBC 来简化了数据持久化逻辑的实现。MyBatis 在其内部使用 JDBC，提供了更简洁的方式实现了数据库操作。

当 MyBatis 将一个 Java 对象作为输入参数执行 INSERT 语句操作时，它会创建一个 PreparedStatement 对象，并且使用 setXXX() 方式对占位符设置相应的参数值。

这里，XXX 可以是 Int，String，Date 等 Java 对象属性类型的任意一个。示例如下：

XML Code

```

1 <insert id="insertStudent" parameterType="Student">
2     INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL,DOB)
3     VALUES(#{studId},#{name},#{email},#{dob})
4 </insert>

```

为执行这个语句，MyBatis 将采取以下一系列动作：

1. 创建一个有占位符的 PreparedStatement 接口，如下：

Java Code

```

1 PreparedStatement pstmt = connection.prepareStatement
2     ("INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL,DOB) VALUES(?,?,?,?)");

```

2. 检查 Student 对象的属性 studId 的类型 然后使用合适 setXXX 方法去设置参数值。这里 studId 是 integer 类型，所以会使用 setInt() 方法：

Java Code

```

1 pstmt.setInt(1, student.getStudId());

```

3. 类似地，对于 name 和 email 属性都是 String 类型，MyBatis 使用 setString() 方法设置参数。

Java Code

```

1 pstmt.setString(2, student.getName());
2 pstmt.setString(3, student.getEmail());

```

4. 至于 dob 属性，MyBatis 会使用 setDate() 方法设置 dob 处占位符位置的值。
5. MyBatis 会将 java.util.Date 类型转换为 java.sql.Timestamp 并设值：

Java Code

```

1 pstmt.setTimestamp(4, new Timestamp((student.getDob()).
2     getTime()));

```

厉害！但 MyBatis 是怎么知道对于 Integer 类型属性使用 setInt() 和 String 类型属性使用 setString() 方法呢？其实 MyBatis 是通过使用类型处理器 (type handlers) 来决定这么做的。

MyBatis 对于以下的类型使用内建的类型处理器：所有的基本数据类型、基本类型的包裹类型、byte[]、java.util.Date、java.sql.Date、java.sql.Time、java.sql.Timestamp、java 枚举类型等。所以当 MyBatis 发现属性的类型属于上述类型，他会使用对应的类型处理器将值设置到 PreparedStatement 中，同样地，当从 SQL 结果集构建 JavaBean 时，也有类似的过程。

那如果我们给了一个自定义的对象类型，来存储存储到数据库呢？示例如下：

假设表 STUDENTS 有一个 PHONE 字段，类型为 VARCHAR(15)，而 JavaBean Student 有一个 PhoneNumber 类定义类型的 phoneNumber 属性。

Java Code

```

1 public class PhoneNumber
2 {
3     private String countryCode;
4     private String stateCode;
5     private String number;
6     public PhoneNumber()
7     {
8     }
9     public PhoneNumber(String countryCode, String stateCode, String
10         number)
11     {
12         this.countryCode = countryCode;
13         this.stateCode = stateCode;
14         this.number = number;
15     }
16     public PhoneNumber(String string)
17     {
18         if(string != null)
19         {
20             String[] parts = string.split("-");
21             if(parts.length > 0) this.countryCode = parts[0];
22             if(parts.length > 1) this.stateCode = parts[1];
23             if(parts.length > 2) this.number = parts[2];
24         }
25     }
26     public String getAsString()
27     {
28         return countryCode + "-" + stateCode + "-" + number;
29     }
30     // Setters and getters
31 }
32
33 public class Student
34 {
35     private Integer id;
36     private String name;
37     private String email;
38     private PhoneNumber phone;
39     // Setters and getters
40 }

```

XML Code

```

1 <insert id="insertStudent" parameterType="Student">
2     insert into students(name,email,phone)
3     values("#{name},#{email},#{phone})
4 </insert>

```

这里，phone 参数需要传递给#{phone}；而 phone 对象是 PhoneNumber 类型。然而，MyBatis 并不知道怎样来处理这个类型的对象。

为了让 MyBatis 明白怎样处理这个自定义的 Java 对象类型，如 PhoneNumber，我们可以创建一个自定义的类型处理器，如下所示：

1. MyBatis 提供了抽象类 BaseTypeHandler<T>，我们可以继承此类创建自定义类型处理器。

Java Code

```

1 package com.mybatis3.typehandlers;
2 import java.sql.CallableStatement;
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5 import java.sql.SQLException;
6 import org.apache.ibatis.type.BaseTypeHandler;
7 import org.apache.ibatis.type.JdbcType;
8 import com.mybatis3.domain.PhoneNumber;
9 public class PhoneTypeHandler extends BaseTypeHandler<PhoneNumber>
10 {
11     @Override
12     public void setNonNullParameter(PreparedStatement ps, int i,
13                                     PhoneNumber parameter, JdbcType jdbcType) throws
14         SQLException
15     {
16         ps.setString(i, parameter.getAsString());
17     }
18     @Override
19     public PhoneNumber getNullableResult(ResultSet rs, String
20                                         columnName)
21     throws SQLException
22     {
23         return new PhoneNumber(rs.getString(columnName));
24     }
25     @Override
26     public PhoneNumber getNullableResult(ResultSet rs, int
27                                         columnIndex)
28     throws SQLException
29     {
30         return new PhoneNumber(rs.getString(columnIndex));

```

```

31     }
32     @Override
33     public PhoneNumber getNullableResult(CallableStatement cs, int
34                                     columnIndex)
35     throws SQLException
36     {
37         return new PhoneNumber(cs.getString(columnIndex));
38     }
39 }

```

2. 我们使用 `ps.setString()` 和 `rs.getString()` 方法是因为 `phone` 列是 `VARCHAR` 类型。

3. 一旦我们实现了自定义的类型处理器，我们需要在 `mybatis-config.xml` 中注册它：

XML Code

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
3  "http://mybatis.org/dtd/mybatis-3-config.dtd">
4  <configuration>
5      <properties resource="application.properties" />
6      <typeHandlers>
7          <typeHandler handler="com.mybatis3.typehandlers. PhoneTypeHandler" />
8      </typeHandlers>
9  </configuration>
10

```

注册 `PhoneTypeHandler` 后，MyBatis 就能够将 `Phone` 类型的对象值存储到 `VARCHAR` 类型的列上。

2.1.7 全局参数设置 Settings

为满足应用特定的需求，MyBatis 默认的全局参数设置可以被覆盖(overridden)掉，如下所示：

XML Code

```

1  <settings>
2      <setting name="cacheEnabled" value="true" />
3      <setting name="lazyLoadingEnabled" value="true" />
4      <setting name="multipleResultSetsEnabled" value="true" />
5      <setting name="useColumnLabel" value="true" />
6      <setting name="useGeneratedKeys" value="false" />
7      <setting name="autoMappingBehavior" value="PARTIAL" />
8      <setting name="defaultExecutorType" value="SIMPLE" />
9      <setting name="defaultStatementTimeout" value="25000" />
10     <setting name="safeRowBoundsEnabled" value="false" />
11     <setting name="mapUnderscoreToCamelCase" value="false" />

```



```

12 <setting name="localCacheScope" value="SESSION" />
13 <setting name="jdbcTypeForNull" value="OTHER" />
14 <setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode ,toString" />
15 </settings>
16

```

2.1.8 SQL 映射定义 Mappers

Mapper XML 文件中包含的 SQL 映射语句将会被应用通过使用其 `statementid` 来执行。我们需要在 `mybatis-config.xml` 文件中配置 SQL Mapper 文件的位置。

XML Code

```

1 <mappers>
2   <mapper resource="com/mybatis3/mappers/StudentMapper.xml" />
3   <mapper url="file:///D:/mybatisdemo/app/mappers/TutorMapper.xml" />
4   <mapper class="com.mybatis3.mappers.TutorMapper" />
5   <package name="com.mybatis3.mappers" />
6 </mappers>
7

```

以上每一个 `<mapper>` 标签的属性有助于从不同类型的资源中加载映射 mapper：

- `resource` 属性用来指定在 `classpath` 中的 mapper 文件。
- `url` 属性用来通过完全文件系统路径或者 web URL 地址来指向 mapper 文件
- `class` 属性用来指向一个 mapper 接口
- `package` 属性用来指向可以找到 Mapper 接口的包名

2.2 使用 Java API 配置 MyBatis

上一节中，我们已经讨论了各种 MyBatis 配置元素，如 `environments`, `typeAlias`, 和 `typeHandlers`，以及如何使用 XML 配置它们。即使你想使用基于 Java API 的 MyBatis 配置，经历上一节的学习是大有好处的，它可以帮你对这些配置元素有更好的理解。在本节中，我们会引用到前一节中描述的一些类。

MyBatis 的 `SqlSessionFactory` 接口除了使用基于 XML 的配置创建外也可以通过 Java API 编程式地被创建。每个在 XML 中配置的元素，都可以编程式的创建。

使用 Java API 创建 `SqlSessionFactory`，代码如下：

Java Code

```

1 public static SqlSessionFactory getSqlSessionFactory()
2 {

```

```

3    SqlSessionFactory sqlSessionFactory = null;
4    try
5    {
6        DataSource dataSource = DataSourceFactory.getDataSource();
7        TransactionFactory transactionFactory = new
8            JdbcTransactionFactory();
9        Environment environment = new Environment("development",
10            transactionFactory, dataSource);
11        Configuration configuration = new Configuration(environment);
12        configuration.getTypeAliasRegistry().registerAlias("student",
13            Student.class);
14        configuration.getTypeHandlerRegistry().register(PhoneNumber.
15            class, PhoneTypeHandler.class);
16        configuration.addMapper(StudentMapper.class);
17        sqlSessionFactory = new SqlSessionFactoryBuilder().
18            build(configuration);
19    }
20    catch (Exception e)
21    {
22        throw new RuntimeException(e);
23    }
24    return sqlSessionFactory;
25 }

```

2.2.1 环境配置 Environment

我们需要为想使用 MaBatis 连接的每一个数据库创建一个 Enviroment 对象。为了使用每一个环境，我们需要为每一个环境 environment 创建一个 SqlSessionFactory 对象。而创建 Environment 对象，我们需要 java.sql.DataSource 和 TransactionFactory 实例。下面让我们看看如何创建 DataSource 和 TransactionFactory 对象。

2.2.2 数据源 DataSource

MyBatis 支持三种内建的 DataSource 类型：UNPOOLED，POOLED，和 JNDI。

- UNPOOLED 类型的数据源 dataSource 为每一个用户请求创建一个数据库连接。在多用户并发应用中，不建议使用。
- POOLED 类型的数据源 dataSource 创建了一个数据库连接池，对用户的每一个请求，会使用缓冲池中的一个可用的 Connection 对象，这样可以提高应用的性能。MyBatis 提供了 org.apache.ibatis.datasource.pooled.PooledDataSource 实现 javax.sql.DataSource 来创建连接池。
- JNDI 类型的数据源 dataSource 使用了应用服务器的数据库连接池，并且使用 JNDI 查找来获取数据库连接。

让我们看一下怎样通过 MyBatis 的 PooledDataSource 获得 DataSource 对象，如下：

Java Code

```
1 public class DataSourceFactory
2 {
3     public static DataSource getDataSource()
4     {
5         String driver = "com.mysql.jdbc.Driver";
6         String url = "jdbc:mysql://localhost:3306/mybatisdemo";
7         String username = "root";
8         String password = "admin";
9         PooledDataSource dataSource = new PooledDataSource(driver, url,
10             username, password);
11         return dataSource;
12     }
13 }
```

一般在生产环境中，DataSource 会被应用服务器配置，并通过 JNDI 获取 DataSource 对象，如下所示：

Java Code

```
1 public class DataSourceFactory
2 {
3     public static DataSource getDataSource()
4     {
5         String jndiName = "java:comp/env/jdbc/MyBatisDemoDS";
6         try
7         {
8             InitialContext ctx = new InitialContext();
9             DataSource dataSource = (DataSource) ctx.lookup(jndiName);
10            return dataSource;
11        }
12        catch (NamingException e)
13        {
14            throw new RuntimeException(e);
15        }
16    }
17 }
```

当前有一些流行的第三方类库，如 commons-dbcp 和 c3p0 实现了 java.sql.DataSource，你可以使用它们来创建 dataSource。

2.2.3 事务工厂 TransactionFactory

MyBatis 支持一下两种 TransactionFactory 实现：

- JdbcTransactionFactory

- ManagedTransactionFactory

如果你的应用程序运行在未托管 (non-managed) 的环境中, 你应该使用 JdbcTransactionFactory。

Java Code

```
1 DataSource dataSource = DataSourceFactory.getDataSource();
2 TransactionFactory txnFactory = new JdbcTransactionFactory();
3 Environment environment = new Environment("development", txnFactory,
4     dataSource);
```

如果你的应用程序运行在未托管 (non-managed) 的环境中, 并且使用容器支持的事务管理服务, 你应该使用 ManagedTransactionFactory。

Java Code

```
1 DataSource dataSource = DataSourceFactory.getDataSource();
2 TransactionFactory txnFactory = new ManagedTransactionFactory();
3 Environment environment = new Environment("development", txnFactory,
4     dataSource);
```

2.2.4 类型别名 typeAliases

MyBatis 提供以下几种通过 Configuration 对象注册类型别名的方法：

1. 根据默认的别名规则, 使用一个类的首字母小写、非完全限定的类名作为别名注册, 可使用以下代码：

Java Code

```
1 configuration.getTypeAliasRegistry().registerAlias(Student.class);
```

2. 指定指定别名注册, 可使用以下代码：

Java Code

```
1 configuration.getTypeAliasRegistry().registerAlias("Student", Student.class);
```

3. 通过类的完全限定名注册相应类别名, 可使用一下代码：

Java Code

```
1 configuration.getTypeAliasRegistry().registerAlias("Student",
2     "com.mybatis3.domain.Student");
```

4. 为某一个包中的所有类注册别名, 可使用以下代码：

Java Code

```
1 configuration.getTypeAliasRegistry().registerAliases("com.
2     mybatis3.domain");
```

5. 为在 com.mybatis3.domain package 包中所有的继承自 Identifiable 类型的类注册别名, 可使用以下代码：

Java Code

```
1 configuration.getTypeAliasRegistry().registerAliases("com.  
2     mybatis3.domain", Identifiable.class);
```

2.2.5 类型处理器 typeHandlers

MyBatis 提供了一系列使用 Configuration 对象注册类型处理器 (type handler) 的方法。我们可以通过以下方式注册自定义的类处理器：

1. 为某个特定的类注册类处理器：

Java Code

```
1 configuration.getTypeHandlerRegistry().register(PhoneNumber.  
2     class, PhoneTypeHandler.class);
```

2. 注册一个类处理器：

Java Code

```
1 configuration.getTypeHandlerRegistry().register(PhoneTypeHandler.  
2     class);
```

3. 注册 com.mybatis3.typehandlers 包中的所有类型处理器：

Java Code

```
1 configuration.getTypeHandlerRegistry().register("com.mybatis3.typehandlers");
```

2.2.6 全局参数设置 Settings

MyBatis 提供了一组默认的，能够很好地适用大部分的应用的全局参数设置。然而，你可以稍微调整这些参数，让它更好地满足你应用的需要。你可以使用下列方法将全局参数设置成想要的值。

Java Code

```
1 configuration.setCacheEnabled(true);  
2 configuration.setLazyLoadingEnabled(false);  
3 configuration.setMultipleResultSetsEnabled(true);  
4 configuration.setUseColumnLabel(true);  
5 configuration.setUseGeneratedKeys(false);  
6 configuration.setAutoMappingBehavior(AutoMappingBehavior.PARTIAL);  
7 configuration.setDefaultExecutorType(ExecutorType.SIMPLE);  
8 configuration.setDefaultStatementTimeout(25);  
9 configuration.setSafeRowBoundsEnabled(false);  
10 configuration.setMapUnderscoreToCamelCase(false);  
11 configuration.setLocalCacheScope(LocalCacheScope.SESSION);  
12 configuration.setAggressiveLazyLoading(true);  
13 configuration.setJdbcTypeForNull(JdbcType.OTHER);  
14 Set<String> lazyLoadTriggerMethods = new HashSet<String>();
```

```

15 lazyLoadTriggerMethods.add("equals");
16 lazyLoadTriggerMethods.add("clone");
17 lazyLoadTriggerMethods.add("hashCode");
18 lazyLoadTriggerMethods.add("toString");
19 configuration.setLazyLoadTriggerMethods(lazyLoadTriggerMethods );
20

```

2.2.7 Mappers

MyBatis 提供了一些使用 Configuration 对象注册 Mapper XML 文件和 Mapper 接口的方法。

1. 添加一个 Mapper 接口，可使用以下代码：

Java Code

```

1 configuration.addMapper(StudentMapper.class);

```

2. 添加 com.mybatis3.mappers 包中的所有 Mapper XML 文件或者 Mapper 接口，可使用以下代码：

Java Code

```

1 configuration.addMappers("com.mybatis3.mappers");

```

3. 添加所有 com.mybatis3.mappers 包中的拓展了特定 Mapper 接口的 Mapper 接口，如 BaseMapper，可使用如下代码：

Java Code

```

1 configuration.addMappers("com.mybatis3.mappers", BaseMapper.class);

```

[ Mappers 应该在 typeAliases 和 typeHandler 注册后再添加到 configuration 中。]

2.3 自定义 MyBatis 日志

MyBatis 使用其内部 LoggerFactory 作为真正的日志类库使用的门面。其内部的 LoggerFactory 会将日志记录任务委托给如下的所示某一个日志实现，日志记录优先级由上到下顺序递减：


- SLF4J
- Apache Commons Logging
- Log4j 2
- Log4j
- JDK logging

如果 MyBatis 未发现上述日志记录实现，则 MyBatis 的日志记录功能无效。

如果你的运行环境中，在 classpath 中有多个可用的日志类库，并且你希望 MyBatis 使用某个特定的日志实现，你可以通过调用以下其中一个方法：

- org.apache.ibatis.logging.LogFactory.useSlf4jLogging();
- org.apache.ibatis.logging.LogFactory.useLog4JLogging();

- `org.apache.ibatis.logging.LogFactory.useLog4J2Logging();`
- `org.apache.ibatis.logging.LogFactory.useJdkLogging();`
- `org.apache.ibatis.logging.LogFactory.useCommonsLogging();`
- `org.apache.ibatis.logging.LogFactory.useStdOutLogging();`

 如果你想自定义 MyBatis 日志记录，你应该在调用任何其它方法之前调用以上的其中一个方法。如果你想切换到的日志记录类库在运行时期无效，MyBatis 将会忽略这一请求。

2.4 总结

在本章中我们学习了怎样使用 XML 配置和基于 Java API 配置的方式引导 MyBatis。我们还学习了各种配置选项，如类型别名 `type alias`，类型处理器 `type handlers`，以及全局参数设置。在接下来的一章里，我们会讨论 SQL Mappers (SQL 映射定义)，它们是 MyBaBatis 的核心元素。

3

第三章 使用 XML 配置 SQL 映射器

关系型数据库和 SQL 是经受时间考验和验证的数据存储机制。和其他的 ORM 框架如 Hibernate 不同，MyBatis 鼓励开发者可以直接使用数据库，而不是将其对开发者隐藏，因为这样可以充分发挥数据库服务器所提供的 SQL 语句的巨大威力。与此同时，MyBatis 消除了书写大量冗余代码的痛苦，它使使用 SQL 更容易。

在代码里直接嵌套 SQL 语句是很差的编码实践，并且维护起来困难。MyBatis 使用了映射器配置文件或注解来配置 SQL 语句。在本章中，我们会看到具体怎样使用映射器配置文件来配置映射 SQL 语句。

本章将涵盖以下话题：

- 映射器配置文件 和 映射器接口
- 映射语句
 - 配置 INSERT, UPDATE, DELETE, and SELECT 语句
- 结果映射 ResultMaps
 - 简单 ResultMaps
 - 使用内嵌 select 语句子查询的一对一映射
 - 使用内嵌的结果集查询的一对一映射
 - 使用内嵌 select 语句子查询的一对多映射
 - 使用内嵌的结果集查询的一对一映射
- 动态 SQL 语句
 - If 条件
 - choose (when, otherwise) 条件
 - trim (where, set) 条件
 - foreach 循环
- MyBatis 菜谱

3.1 映射器配置文件和映射器接口

在前几章中,我们已经看见了一些在映射器配置文件中配置基本的映射语句,以及怎样使用 `SqlSession` 对象调用它们的例子。

现在让我们看一下在 `com.mybatis3.mappers` 包中的 `StudentMapper.xml` 配置文件内,是如何配置 `id` 为“`findStudentById`”的 SQL 语句的,代码如下:

XML Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4 <mapper namespace="com.mybatis3.mappers.StudentMapper">
5   <select id="findStudentById" parameterType="int" resultType="Student">
6     select stud_id as studId, name, email, dob
7     from Students where stud_id=#{studId}
8   </select>
9 </mapper>
10
```

我们可以通过下列代码调用 `findStudentById` 映射的 SQL 语句:

Java Code

```
1 public Student findStudentById(Integer studId)
2 {
3     SqlSession sqlSession = MyBatisUtil.getSqlSession();
4     try
5     {
6         Student student =
7             sqlSession.selectOne("com.mybatis3.mappers.StudentMapper.
8                                 findStudentById", studId);
9         return student;
10    }
11    finally
12    {
13        sqlSession.close();
14    }
15 }
```

我们可以通过字符串(字符串形式为:映射器配置文件所在的包名 `namespace` + 在文件内定义的语句 `id`,如上,即包名 `com.mybatis3.mappers.StudentMapper` 和语句 `id findStudentById` 组成)调用映射的 SQL 语句,但是这种方式容易出错。你需要检查映射器配置文件中的定义,以保证你的输入参数类型和结果返回类型是有效的。

MyBatis 通过使用映射器 Mapper 接口提供了更好的调用映射语句的方法。一旦我们通过映射器配置文件配置了映射语句，我们可以创建一个完全对应的一个映射器接口，接口名跟配置文件名相同，接口所在包名也跟配置文件所在包名完全一样（如 StudentMapper.xml 所在的包名是 com.mybatis3.mappers，对应的接口名就是 com.mybatis3.mappers.StudentMapper.java）。映射器接口中的方法签名也跟映射器配置文件中完全对应：方法名为配置文件中 id 值；方法参数类型为 parameterType 对应值；方法返回值类型为 returnType 对应值。

对于上述的 StudentMapper.xml 文件，我们可以创建一个映射器接口 StudentMapper.java 如下：

Java Code

```
1 package com.mybatis3.mappers;
2 public interface StudentMapper
3 {
4     Student findStudentById(Integer id);
5 }
```

在 StudentMapper.xml 映射器配置文件中，其 namespace 应该跟 StudentMapper 接口的完全限定名保持一致。另外，StudentMapper.xml 中语句 id, parameterType, returnType 应该分别和 StudentMapper 接口中的方法名，参数类型，返回值相对应。

使用映射器接口我们可以以类型安全的形式调用调用映射语句。如下所示：

Java Code

```
1 public Student findStudentById(Integer studId)
2 {
3     SqlSession sqlSession = MyBatisUtil.getSqlSession();
4     try
5     {
6         StudentMapper studentMapper =
7             sqlSession.getMapper(StudentMapper.class);
8         return studentMapper.findStudentById(studId);
9     }
10    finally
11    {
12        sqlSession.close();
13    }
14 }
```



即使映射器 Mapper 接口可以以类型安全的方式调用映射语句，但是我们负责书写正确的，匹配方法名、参数类型、和返回值的映射器 Mapper 接口。如果映射器 Mapper 接口中的方法和 XML 中的映射语句不能匹配，会在运行期抛出一个异常。实际上，指定 `parameterType` 是可选的；MyBatis 可以使用反射机制来决定 `parameterType`。但是，从配置可读性的角度来看，最好指定 `parameterType` 属性。如果 `parameterType` 没有被提及，开发者必须查看 Mapper XML 配置和 Java 代码了解传递给语句的输入参数的数据类型。

3.2 映射语句

MyBatis 提供了多种元素来配置不同类型的语句，如 SELECT，INSERT，UPDATE，DELETE。接下来让我们看看如何具体配置映射语句

3.2.1 INSERT 语句

一个 INSERT SQL 语句可以在 `<insert>` 元素在映射器 XML 配置文件中配置，如下所示：

XML Code

```
1 <insert id="insertStudent" parameterType="Student">
2     INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL, PHONE)
3     VALUES(#{studId},#{name},#{email},#{phone})
4 </insert>
```

这里我们使用一个 ID `insertStudent`，可以在名空间 `com.mybatis3.mappers.StudentMapper.insertStudent` 中唯一标识。`parameterType` 属性应该是一个完全限定类名或者是一个类型别名（`alias`）。

我们可以如下调用这个语句：

Java Code

```
1 int count =
2 sqlSession.insert("com.mybatis3.mappers.StudentMapper.insertStudent", student);
```

`sqlSession.insert()` 方法返回执行 INSERT 语句后所影响的行数。

如果不使用名空间（`namespace`）和语句 `id` 来调用映射语句，你可以通过创建一个映射器 Mapper 接口，并以类型安全的方式调用方法，如下所示：

Java Code

```
1 package com.mybatis3.mappers;
2 public interface StudentMapper
```

```

3 {
4     int insertStudent(Student student);
5 }

```

你可以如下调用 insertStudent 映射语句：

Java Code

```

1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 int count = mapper.insertStudent(student);

```

[自动生成主键]

在上述的 INSERT 语句中，我们为可以自动生成（auto-generated）主键的列 STUD_ID 插入值。我们可以使用 useGeneratedKeys 和 keyProperty 属性让数据库生成 auto_increment 列的值，并将生成的值设置到其中一个输入对象属性内，如下所示：

XML Code

```

1 <insert id="insertStudent" parameterType="Student" useGeneratedKeys="true"
2     keyProperty="studId">
3     INSERT INTO STUDENTS(NAME, EMAIL, PHONE)
4     VALUES(#{name},#{email},#{phone})
5 </insert>
6

```

这里 STUD_ID 列值将会被 MySQL 数据库自动生成，并且生成的值会被设置到 student 对象的 studId 属性上。

Java Code

```

1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 mapper.insertStudent(student);

```

现在可以如下获取插入的 STUDENT 记录的 STUD_ID 的值：

Java Code

```

1 int studentId = student.getStudId();

```

有些数据库如 Oracle 并不支持 AUTO_INCREMENT 列，其使用序列（SEQUENCE）来生成主键值。假设我们有一个名为 STUD_ID_SEQ 的序列来生成 SUTD_ID 主键值。使用如下代码来生成主键：

XML Code

```
1 <insert id="insertStudent" parameterType="Student">
2   <selectKey keyProperty="studId" resultType="int" order="BEFORE">
3     SELECT EARNING.STUD_ID_SEQ.NEXTVAL FROM DUAL
4   </selectKey>
5   INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL, PHONE)
6     VALUES(#{studId},#{name},#{email},#{phone})
7 </insert>
```

这里我们使用了<selectKey>子元素来生成主键值，并将值保存到 Student 对象的 studId 属性上。属性 order="before"表示 MyBatis 将取得序列的下一个值作为主键值，并且在执行 INSERT SQL 语句之前将值设置到 studId 属性上。

我们也可以在获取序列的下一个值时，使用触发器 (trigger) 来设置主键值，并且在执行 INSERT SQL 语句之前将值设置到主键列上。如果你采取这样的方式，则对应的 INSERT 映射语句如下所示：

XML Code

```
1 <insert id="insertStudent" parameterType="Student">
2   INSERT INTO STUDENTS(NAME,EMAIL, PHONE)
3     VALUES(#{name},#{email},#{phone})
4   <selectKey keyProperty="studId" resultType="int" order="AFTER">
5     SELECT EARNING.STUD_ID_SEQ.CURRVAL FROM DUAL
6   </selectKey>
7 </insert>
```

3.2.2 UPDATE 语句

一个 UPDATE SQL 语句可以在<update>元素在映射器 XML 配置文件中配置，如下所示：

XML Code

```
1 <update id="updateStudent" parameterType="Student">
2   UPDATE STUDENTS SET NAME=#{name}, EMAIL=#{email}, PHONE=#{phone}
3   WHERE STUD_ID=#{studId}
4 </update>
```

我们可以如下调用此语句：

Java Code

```
1 int noOfRowsUpdated =
2 sqlSession.update("com.mybatis3.mappers.StudentMapper.updateStudent", student);
```

sqlSession.update() 方法返回执行 UPDATE 语句之后影响的行数。

如果不使用名空间 (namespace) 和语句 id 来调用映射语句，你可以通过创建一个映射器 Mapper 接口，并以类型安全的方式调用方法，如下所示：

Java Code

```
1 package com.mybatis3.mappers;
2 public interface StudentMapper
3 {
4     int updateStudent(Student student);
5 }
```

你可以使用映射器 Mapper 接口来调用 updateStudent 语句，如下所示：

Java Code

```
1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 int noOfRowsUpdated = mapper.updateStudent(student);
```

3.2.3 删除语句

一个 UPDATE SQL 语句可以在<update>元素在映射器 XML 配置文件中配置，如下所示：

XML Code

```
1 <delete id="deleteStudent" parameterType="int">
2     DELETE FROM STUDENTS WHERE STUD_ID=#{studId}
3 </delete>
```

我们可以如下调用此语句：

Java Code

```
1 int studId = 1;
2 int noOfRowsDeleted =
3     sqlSession.delete("com.mybatis3.mappers.StudentMapper.deleteStudent", studId);
```

sqlSession.delete() 方法返回 delete 语句执行后影响的行数。

如果不使用名空间 (namespace) 和语句 id 来调用映射语句，你可以通过创建一个映射器 Mapper 接口，并以类型安全的方式调用方法，如下所示：

Java Code

```
1 package com.mybatis3.mappers;
2 public interface StudentMapper
3 {
```

```

4     int deleteStudent(int studId);
5 }

```

你可以使用映射器 Mapper 接口来调用 updateStudent 语句，如下所示：

Java Code

```

1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 int noOfRowsDeleted = mapper.deleteStudent(studId);

```

3.2.4 SELECT 语句

MyBatis 真正强大的功能，在于映射 SELECT 查询结果到 JavaBeans 方面的极大灵活性。

让我们看看一个简单的 select 查询是如何（在 MyBatis 中）配置的，如下所示：

XML Code

```

1 <select id="findStudentById" parameterType="int"
2     resultType="Student">
3     SELECT STUD_ID, NAME, EMAIL, PHONE
4     FROM STUDENTS
5     WHERE STUD_ID=#{studId}
6 </select>

```

我们可以如下调用此语句：

Java Code

```

1 int studId =1;
2 Student student = sqlSession.selectOne("com.mybatis3.mappers.
3 StudentMapper.findStudentById", studId);

```

如果不使用名空间（namespace）和语句 id 来调用映射语句，你可以通过创建一个映射器 Mapper 接口，并以类型安全的方式调用方法，如下所示：

Java Code

```

1 package com.mybatis3.mappers;
2 public interface StudentMapper
3 {
4     Student findStudentById(Integer studId);
5 }

```

你可以使用映射器 Mapper 接口来调用 updateStudent 语句，如下所示：

Java Code

```

1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 Student student = mapper.findStudentById(studId);

```

如果你检查 Student 对象的属性值，你会发现 studId 属性值并没有被 stud_id 列值填充。这是因为 MyBatis 自动对 JavaBean 中和列名匹配的属性进行填充。这就是为什么 name ,email,和 phone 属性被填充，而 studId 属性没有被填充。

为了解决这一问题，我们可以为列名起一个可以与 JavaBean 中属性名匹配的别名，如下所示：

XML Code

```

1 <select id="findStudentById" parameterType="int"
2   resultType="Student">
3     SELECT STUD_ID AS studId, NAME,EMAIL, PHONE
4     FROM STUDENTS
5     WHERE STUD_ID=#{studId}
6 </select>

```

现在，Student 这个 Bean 对象中的值将会恰当地被 stud_id,name,email,phone 列填充了。

现在，让我们看一下如何执行返回多条结果的 SELECT 语句查询，如下所示：

XML Code

```

1 <select id="findAllStudents" resultType="Student">
2     SELECT STUD_ID AS studId, NAME,EMAIL, PHONE
3     FROM STUDENTS
4 </select>

```

XML Code

```

1 List<Student> students =
2 sqlSession.selectList("com.mybatis3.mappers.StudentMapper.findAllStudents");

```

映射器 Mapper 接口 StudentMapper 可以如下定义：

Java Code

```

1 package com.mybatis3.mappers;
2 public interface StudentMapper
3 {
4     List<Student> findAllStudents();
5 }

```

使用上述代码，我们可以如下调用 findAllStudents 语句：

Java Code


```

1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 List<Student> students = mapper.findAllStudents();

```

如果你注意到上述的 SELECT 映射定义，你可以看到，我们为所有的映射语句中的 stud_id 起了别名。

我们可以使用 ResultMaps，来避免上述的到处重复别名。我们稍后会继续讨论。

除了 java.util.List，你也可以是由其他类型的集合类，如 Set, Map，以及 (SortedSet)。MyBatis 根据集合的类型，会采用适当的集合实现，如下所示：

- 对于 List, Collection, Iterable 类型，MyBatis 将返回 java.util.ArrayList
- 对于 Map 类型，MyBatis 将返回 java.util.HashMap
- 对于 Set 类型，MyBatis 将返回 java.util.HashSet
- 对于 SortedSet 类型，MyBatis 将返回 java.util.TreeSet

3.3 结果集映射 ResultMaps

ResultMaps 被用来 将 SQL SELECT 语句的结果集映射到 JavaBeans 的属性中。我们可以定义结果集映射 ResultMaps 并且在一些 SELECT 语句上引用 resultMap。MyBatis 的结果集映射 ResultMaps 特性非常强大，你可以使用它将简单的 SELECT 语句映射到复杂的一对一和一对多关系的 SELECT 语句上。

3.3.1 简单 resultMap

一个映射了查询结果和 Student JavaBean 的简单的 resultMap 定义如下：

XML Code

```

1 <resultMap id="StudentResult" type="com.mybatis3.domain.Student">
2   <id property="studId" column="stud_id" />
3   <result property="name" column="name" />
4   <result property="email" column="email" />
5   <result property="phone" column="phone" />
6 </resultMap>
7
8 <select id="findAllStudents" resultMap="StudentResult">
9   SELECT * FROM STUDENTS
10 </select>
11
12 <select id="findStudentById" parameterType="int" resultMap="StudentResult">
13   SELECT * FROM STUDENTS WHERE STUD_ID=#{studId}
14 </select>
15

```

表示 resultMap 的 StudentResult id 值应该在此名空间内是唯一的。并且 type 属性应该是完全限定类名或者是返回类型的别名。

<result>子元素被用来将一个 resultSet 列映射到 JavaBean 的一个属性中。

<id>元素和<result>元素功能相同 不过它被用来映射到唯一标识属性 ,用来区分和比较对象(一般和主键列相对应)。
在<select>语句中,我们使用了 resultMap 属性,而不是 resultType 来引用 StudentResult 映射。当<select>语句中配置了 resultMap 属性,MyBatis 会使用此数据库列名与对象属性映射关系来填充 JavaBean 中的属性。



resultType 和 resultMap 二者只能用其一,不能同时使用。

让我们来看另外一个<select>映射语句定义的例子,怎样将查询结果填充到 HashMap 中。如下所示:

XML Code

```
1 <select id="findStudentById" parameterType="int" resultType="map">
2     SELECT * FROM STUDENTS WHERE STUD_ID=#{studId}
3 </select>
```

在上述的<select>语句中,我们将 resultType 配置成 map,即 java.util.HashMap 的别名。在这种情况下,结果集的列名将会作为 Map 中的 key 值,而列值将作为 Map 的 value 值。

Java Code

```
1 HashMap<String, Object> studentMap = sqlSession.selectOne("com.
2 mybatis3.mappers.StudentMapper.findStudentById", studId);
3 System.out.println("stud_id :"+studentMap.get("stud_id"));
4 System.out.println("name :"+studentMap.get("name"));
5 System.out.println("email :"+studentMap.get("email"));
6 System.out.println("phone :"+studentMap.get("phone"));
```

让我们再看一个 使用 resultType="map",返回多行结果的例子:

XML Code

```
1 <select id="findAllStudents" resultType="map">
2     SELECT STUD_ID, NAME, EMAIL, PHONE FROM STUDENTS
3 </select>
```

由于 resultType="map"和语句返回多行,则最终返回的数据类型应该是 List<HashMap<String, Object>>,如下所示:

Java Code

```
1 List<HashMap<String, Object>> studentMapList =
2     sqlSession.selectList("com.mybatis3.mappers.StudentMapper.findAllS
3         tudents");
4 for(HashMap<String, Object> studentMap : studentMapList)
5 {
```

```

6     System.out.println("studId :" + studentMap.get("stud_id"));
7     System.out.println("name :" + studentMap.get("name"));
8     System.out.println("email :" + studentMap.get("email"));
9     System.out.println("phone :" + studentMap.get("phone"));
10 }

```

3.3.2 拓展 resultMap

我们可以从另外一个<resultMap>，拓展出一个新的<resultMap>，这样，原先的属性映射可以继承过来，以实现。

XML Code

```

1 <resultMap type="Student" id="StudentResult">
2   <id property="studId" column="stud_id" />
3   <result property="name" column="name" />
4   <result property="email" column="email" />
5   <result property="phone" column="phone" />
6 </resultMap>
7 <resultMap type="Student" id="StudentWithAddressResult" extends="StudentResult">
8   <result property="address.addrId" column="addr_id" />
9   <result property="address.street" column="street" />
10  <result property="address.city" column="city" />
11  <result property="address.state" column="state" />
12  <result property="address.zip" column="zip" />
13  <result property="address.country" column="country" />
14 </resultMap>
15

```

id 为 StudentWithAddressResult 的 resultMap 拓展了 id 为 StudentResult 的 resultMap。

如果你只想映射 Student 数据，你可以使用 id 为 StudentResult 的 resultMap，如下所示：

XML Code

```

1 <select id="findStudentById" parameterType="int"
2   resultMap="StudentResult">
3   SELECT * FROM STUDENTS WHERE STUD_ID=#{studId}
4 </select>

```

如果你想将映射 Student 数据和 Address 数据，你可以使用 id 为 StudentWithAddressResult 的 resultMap：

XML Code

```

1 <select id="selectStudentWithAddress" parameterType="int"
2 resultMap="StudentWithAddressResult">
3 SELECT STUD_ID, NAME, EMAIL, PHONE, A.ADDR_ID, STREET, CITY,
4       STATE, ZIP, COUNTRY
5 FROM STUDENTS S LEFT OUTER JOIN ADDRESSES A ON
6       S.ADDR_ID=A.ADDR_ID
7 WHERE STUD_ID=#{studId}
8 </select>

```

3.4 一对一映射

在我们的域模型样例中，每一个学生都有一个与之关联的地址信息。表 STUDENTS 有一个 ADDR_ID 列，是 ADDRESSES 表的外键。

STUDENTS 表的样例数据如下所示：

STUD_ID	NAME	EMAIL	PHONE	ADDR_ID
1	John	john@gmail.com	123-456-7890	1
2	Paul	paul@gmail.com	111-222-3333	2

ADDRESSES 表的样例输入如下所示：

ADDR_ID	STREET	CITY	STATE	ZIP	COUNTRY
1	Naperville	CHICAGO	IL	60515	USA
2	Paul	CHICAGO	IL	60515	USA

下面让我们看一下怎样取 Student 明细和其 Address 明细。

Student 和 Address 的 JavaBean 以及映射器 Mapper XML 文件定义如下所示：

Java Code

```

1 public class Address
2 {
3     private Integer addrId;
4     private String street;
5     private String city;
6     private String state;
7     private String zip;
8     private String country;
9     // setters & getters
10 }
11 public class Student
12 {
13     private Integer studId;

```

```

14     private String name;
15     private String email;
16     private PhoneNumber phone;
17     private Address address;
18     //setters & getters
19 }

```

XML Code

```

1 <resultMap type="Student" id="StudentWithAddressResult">
2   <id property="studId" column="stud_id" />
3   <result property="name" column="name" />
4   <result property="email" column="email" />
5   <result property="phone" column="phone" />
6   <result property="address.addrId" column="addr_id" />
7   <result property="address.street" column="street" />
8   <result property="address.city" column="city" />
9   <result property="address.state" column="state" />
10  <result property="address.zip" column="zip" />
11  <result property="address.country" column="country" />
12 </resultMap>
13
14 <select id="selectStudentWithAddress" parameterType="int"
15 resultMap="StudentWithAddressResult">
16     SELECT STUD_ID, NAME, EMAIL, A.ADDR_ID, STREET, CITY, STATE,
17           ZIP, COUNTRY
18     FROM STUDENTS S LEFT OUTER JOIN ADDRESSES A ON
19           S.ADDR_ID=A.ADDR_ID
20     WHERE STUD_ID=#{studId}
21 </select>

```

我们可以使用圆点记法为内嵌的对象的属性赋值。在上述的 resultMap 中，Student 的 address 属性使用了圆点记法被赋上了 address 对应列的值。同样地，我们可以访问任意深度的内嵌对象的属性。我们可以如下访问内嵌对象属性：

Java Code

```

1 //接口定义
2 public interface StudentMapper
3 {
4     Student selectStudentWithAddress(int studId);
5 }
6
7
8 //使用
9 int studId = 1;

```

```

10 StudentMapper studentMapper =
11     sqlSession.getMapper(StudentMapper.class);
12 Student student = studentMapper.selectStudentWithAddress(studId);
13 System.out.println("Student :" + student);
14 System.out.println("Address :" + student.getAddress());

```

上述样例展示了一对一关联映射的一种方法。然而，使用这种方式映射，如果 address 结果需要在其他的 SELECT 映射语句中映射成 Address 对象，我们需要为每一个语句重复这种映射关系。MyBatis 提供了更好地实现一对一关联映射的方法：嵌套结果 ResultMap 和嵌套 select 查询语句。接下来，我们将讨论这两种方式。

3.4.1 使用嵌套结果 ResultMap 实现一对一关系映射

我们可以使用一个嵌套结果 ResultMap 方式来获取 Student 及其 Address 信息，代码如下：

XML Code

```

1 <resultMap type="Address" id="AddressResult">
2   <id property="addrId" column="addr_id" />
3   <result property="street" column="street" />
4   <result property="city" column="city" />
5   <result property="state" column="state" />
6   <result property="zip" column="zip" />
7   <result property="country" column="country" />
8 </resultMap>
9 <resultMap type="Student" id="StudentWithAddressResult">
10  <id property="studId" column="stud_id" />
11  <result property="name" column="name" />
12  <result property="email" column="email" />
13  <association property="address" resultMap="AddressResult" />
14 </resultMap>
15
16 <select id="findStudentWithAddress" parameterType="int"
17 resultMap="StudentWithAddressResult">
18     SELECT STUD_ID, NAME, EMAIL, A.ADDR_ID, STREET, CITY, STATE,
19     ZIP, COUNTRY
20     FROM STUDENTS S LEFT OUTER JOIN ADDRESSES A ON
21     S.ADDR_ID=A.ADDR_ID
22     WHERE STUD_ID=#{studId}
23 </select>

```

元素<association>被用来导入“有一个”(has-one)类型的关联。在上述的例子中，我们使用了<association>元素引用了另外的在同一个 XML 文件中定义的<resultMap>。

我们也可以使用<association 定义内联的 resultMap，代码如下所示：

XML Code

```
1 <resultMap type="Student" id="StudentWithAddressResult">
2   <id property="studId" column="stud_id" />
3   <result property="name" column="name" />
4   <result property="email" column="email" />
5   <association property="address" javaType="Address">
6     <id property="addrId" column="addr_id" />
7     <result property="street" column="street" />
8     <result property="city" column="city" />
9     <result property="state" column="state" />
10    <result property="zip" column="zip" />
11    <result property="country" column="country" />
12  </association>
13 </resultMap>
14
```

使用嵌套结果 resultMap 方式，关联的数据可以通过简单的查询语句（如果需要的话，需要与 joins 连接操作配合）进行加载。

3.4.2 使用嵌套查询实现一对一关系映射

我们可以通过使用嵌套 select 查询来获取 Student 及其 Address 信息，代码如下：

XML Code

```
1 <resultMap type="Address" id="AddressResult">
2   <id property="addrId" column="addr_id" />
3   <result property="street" column="street" />
4   <result property="city" column="city" />
5   <result property="state" column="state" />
6   <result property="zip" column="zip" />
7   <result property="country" column="country" />
8 </resultMap>
9
10 <select id="findAddressById" parameterType="int"
11   resultMap="AddressResult">
12   SELECT * FROM ADDRESSES WHERE ADDR_ID=#{id}
13 </select>
14
15 <resultMap type="Student" id="StudentWithAddressResult">
16   <id property="studId" column="stud_id" />
```

```

17 <result property="name" column="name" />
18 <result property="email" column="email" />
19 <association property="address" column="addr_id" select="findAddressById" />
20 </resultMap>
21
22 <select id="findStudentWithAddress" parameterType="int"
23 resultMap="StudentWithAddressResult">
24     SELECT * FROM STUDENTS WHERE STUD_ID=#{Id}
25 </select>

```

在此方式中，<association>元素的 select 属性被设置成了 id 为 findAddressById 的语句。这里，两个分开的 SQL 语句将会在数据库中执行，第一个调用 findStudentById 加载 student 信息，而第二个调用 findAddressById 来加载 address 信息。

Addr_id 列的值将会被作为输入参数传递给 selectAddressById 语句。

我们可以如下调用 findStudentWithAddress 映射语句：

Java Code

```

1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 Student student = mapper.selectStudentWithAddress(studId);
3 System.out.println(student);
4 System.out.println(student.getAddress());

```

3.5 一对多映射

在我们的域模型样例中，一个讲师可以教授一个或者多个课程。这意味着讲师和课程之间存在一对多的映射关系。

我们可以使用<collection>元素将 一对多类型的结果 映射到 一个对象集合上。

TUTORS 表的样例数据如下：

TUTOR_ID	NAME	EMAIL	PHONE	ADDR_ID
1	John	john@gmail.com	123-456-7890	1
2	Ying	ying@gmail.com	111-222-3333	2

COURSE 表的样例数据如下：

COURSE_ID	NAME	DESCRIPTION	START_DATE	END_DATE	TUTOR_ID
1	JavaSE	Java SE	2013-01-10	2013-02-10	1
2	JavaEE	Java EE 6	2013-01-10	2013-03-10	2
3	MyBatis	MyBatis	2013-01-10	2013-02-20	2

在上述的表数据中，John 讲师教授一个课程，而 Ying 讲师教授两个课程。

Course 和 Tutor 的 JavaBean 定义如下：

Java Code

```
1 public class Course
2 {
3     private Integer courseId;
4     private String name;
5     private String description;
6     private Date startDate;
7     private Date endDate;
8     private Integer tutorId;
9     //setters & getters
10 }
11 public class Tutor
12 {
13     private Integer tutorId;
14     private String name;
15     private String email;
16     private Address address;
17     private List<Course> courses;
18     / setters & getters
19 }
```

现在让我们看看如何获取讲师信息以及其所教授的课程列表信息。

<collection>元素被用来将多行课程结果映射成一个课程 Course 对象的一个集合。和一对一映射一样，我们可以使用**嵌套结果 resultMap**和**嵌套 Select 语句**两种方式映射实现一对多映射。

3.5.1 使用内嵌结果 resultMap 实现一对多映射

我们可以使用嵌套结果 resultMap 方式获得讲师及其课程信息，代码如下：

XML Code

```
1 <resultMap type="Course" id="CourseResult">
2   <id column="course_id" property="courseId" />
3   <result column="name" property="name" />
4   <result column="description" property="description" />
5   <result column="start_date" property="startDate" />
6   <result column="end_date" property="endDate" />
7 </resultMap>
8 <resultMap type="Tutor" id="TutorResult">
9   <id column="tutor_id" property="tutorId" />
```

```

10 <result column="tutor_name" property="name" />
11 <result column="email" property="email" />
12 <collection property="courses" resultMap="CourseResult" />
13 </resultMap>
14
15 <select id="findTutorById" parameterType="int"
16 resultMap="TutorResult">
17 SELECT T.TUTOR_ID, T.NAME AS TUTOR_NAME, EMAIL, C.COURSE_ID,
18 C.NAME, DESCRIPTION, START_DATE, END_DATE
19 FROM TUTORS T LEFT OUTER JOIN ADDRESSES A ON T.ADDR_ID=A.ADDR_ID
20 LEFT OUTER JOIN COURSES C ON T.TUTOR_ID=C.TUTOR_ID
21 WHERE T.TUTOR_ID=#{tutorId}
22 </select>

```

这里我们使用了一个简单的使用了 JOINS 连接的 Select 语句获取讲师及其所教课程信息。<collection>元素的 resultMap 属性设置成了 CourseResult，CourseResult 包含了 Course 对象属性与表列名之间的映射。

3.5.2 使用嵌套 Select 语句实现一对多映射

我们可以使用嵌套 Select 语句方式获得讲师及其课程信息，代码如下：

XML Code

```

1 <resultMap type="Course" id="CourseResult">
2   <id column="course_id" property="courseId" />
3   <result column="name" property="name" />
4   <result column="description" property="description" />
5   <result column="start_date" property="startDate" />
6   <result column="end_date" property="endDate" />
7 </resultMap>
8
9 <resultMap type="Tutor" id="TutorResult">
10   <id column="tutor_id" property="tutorId" />
11   <result column="tutor_name" property="name" />
12   <result column="email" property="email" />
13   <association property="address" resultMap="AddressResult" />
14   <collection property="courses" column="tutor_id" select="findCoursesByTutor" />
15 </resultMap>
16
17 <select id="findTutorById" parameterType="int" resultMap="TutorResult">
18   SELECT T.TUTOR_ID, T.NAME AS TUTOR_NAME, EMAIL
19   FROM TUTORS T WHERE T.TUTOR_ID=#{tutorId}
20 </select>
21
22 <select id="findCoursesByTutor" parameterType="int" resultMap="CourseResult">

```

```

21 SELECT * FROM COURSES WHERE TUTOR_ID=#{tutorId}
22 </select>
23
24

```

在这种方式中，<aossication>元素的 select 属性被设置为 id 为 findCourseByTutor 的语句，用来触发单独的 SQL 查询加载课程信息。tutor_id 这一列值将会作为输入参数传递给 findCouresByTutor 语句。

Java Code

```

1 public interface TutorMapper
2 {
3     Tutor findTutorById(int tutorId);
4 }
5 TutorMapper mapper = sqlSession.getMapper(TutorMapper.class);
6 Tutor tutor = mapper.findTutorById(tutorId);
7 System.out.println(tutor);
8 List<Course> courses = tutor.getCourses();
9 for (Course course : courses)
10 {
11     System.out.println(course);
12 }

```



嵌套 Select 语句查询会导致 N+1 选择问题。首先，主查询将会执行（1 次），对于主查询返回的每一行，另外一个查询将会被执行（主查询 N 行，则此查询 N 次）。对于大型数据库而言，这会导致很差的性能问题。

3.6 动态 SQL

有时候，静态的 SQL 语句并不能满足应用程序的需求。我们可以根据一些条件，来动态地构建 SQL 语句。

例如，在 Web 应用程序中，有可能有一些搜索界面，需要输入一个或多个选项，然后根据这些已选择的条件去执行检索操作。在实现这种类型的搜索功能，我们可能需要根据这些条件 来构建动态的 SQL 语句。如果用户提供了任何输入条件，我们需要将那个条件 添加到 SQL 语句的 WHERE 子句中。

MyBatis 通过使用<if>,<choose>,<where>,<foreach>,<trim>元素提供了对构造动态 SQL 语句的高级别支持。

3.6.1 If 条件

<if>元素被用来有条件地嵌入 SQL 片段，如果测试条件被赋值为 true，则相应地 SQL 片段将会被添加到 SQL 语句中。

假定我们有一个课程搜索界面，设置了 **讲师 (Tutor)** 下拉列表框，**课程名称 (CourseName)** 文本输入框，**开始时间 (StartDate)** 输入框，**结束时间 (EndDate)** 输入框，作为搜索条件。假定课讲师下拉列表是必须选的，其他的都是可选的。

当用户点击 搜索 按钮时，我们需要显示符合以下条件的成列表：

- 特定讲师的课程
- 课程名 包含输入的课程名称关键字的课程；如果课程名称输入为空，则取所有课程
- 在开始时间和结束时间段内的课程

我们可以对应的映射语句，如下所示：

XML Code

```
1 <resultMap type="Course" id="CourseResult">
2   <id column="course_id" property="courseId" />
3   <result column="name" property="name" />
4   <result column="description" property="description" />
5   <result column="start_date" property="startDate" />
6   <result column="end_date" property="endDate" />
7 </resultMap>
8
9 <select id="searchCourses" parameterType="hashmap" resultMap="CourseResult"></select>
10     SELECT * FROM COURSES
11         WHERE TUTOR_ID= #{tutorId}
12         <if test="courseName != null">
13             AND NAME LIKE #{courseName}
14         </if>
15         <if test="startDate != null">
16             AND START_DATE >= #{startDate}
17         </if>
18         <if test="endDate != null">
19             AND END_DATE <= #{endDate}
20         </if>
21 </select>
22
```

Java Code

```
1 public interface CourseMapper
2 {
3     List<Course> searchCourses(Map<String, Object> map);
4 }
5 public void searchCourses()
6 {
7     Map<String, Object> map = new HashMap<String, Object>();
8     map.put("tutorId", 1);
9     map.put("courseName", "%java%");
10    map.put("startDate", new Date());
11    CourseMapper mapper = sqlSession.getMapper(CourseMapper.class);
12    List<Course> courses = mapper.searchCourses(map);

```

```

13     for (Course course : courses)
14     {
15         System.out.println(course);
16     }
17 }

```

此处将生成查询语句 `SELECT * FROM COURSES WHERE TUTOR_ID= ? AND NAME like ? AND START_DATE >= ?`。
准备根据给定条件的动态 SQL 查询将会派上用场。



MyBatis 是使用 ONGL (Object Graph Navigation Language) 表达式来构建动态 SQL 语句。



3.6.2 choose,when 和 otherwise 条件

有时候，查询功能是以查询类别为基础的。首先，用户需要选择是否希望通过选择 讲师，课程名称，开始时间，或结束时间作为查询条件类别来进行查询，然后根据选择的查询类别，输入相应的参数。在这样的场景中，我们需要只使用其中一种查询类别。

MyBatis 提供了<choose>元素支持此类型的 SQL 预处理。

现在让我们书写一个适用此情景的 SQL 映射语句。如果没有选择查询类别，则查询开始时间在今天之后的课程，代码如下：

XML Code

```

1 <select id="searchCourses" parameterType="hashmap" resultMap="CourseResult">
2     SELECT * FROM COURSES
3     <choose>
4         <when test="searchBy == 'Tutor'">
5             WHERE TUTOR_ID= #{tutorId}
6         </when>
7         <when test="searchBy == 'CourseName'">
8             WHERE name like #{courseName}
9         </when>
10        <otherwise>
11            WHERE TUTOR start_date >= now()
12        </otherwise>
13    </choose>
14 </select>

```

MyBatis 计算<choose>测试条件的值，且使用第一个值为 TRUE 的子句。如果没有条件为 true 则使用<otherwise>内的子句。

3.6.3 Where 条件

有时候,所有的查询条件(criteria)应该是可选的。在需要使用至少一种查询条件的情况下,我们应该使用 WHERE 子句。并且, 如果有多个条件,我们需要在条件中添加 AND 或 OR。MyBatis 提供了<where>元素支持这种类型的动态 SQL 语句。

在我们查询课程界面,我们假设所有的查询条件是可选的。进而,当需要提供一个或多个查询条件时,应该改使用 WHERE 子句。

XML Code

```
1 <select id="searchCourses" parameterType="hashmap"
2 resultMap="CourseResult">
3     SELECT * FROM COURSES
4     <where>
5         <if test=" tutorId != null ">
6             TUTOR_ID= #{tutorId}
7         </if>
8         <if test="courseName != null">
9             AND name like #{courseName}
10        </if>
11        <if test="startDate != null">
12            AND start_date >= #{startDate}
13        </if>
14        <if test="endDate != null">
15            AND end_date <= #{endDate}
16        </if>
17    </where>
18 </select>
```

<where>元素只有在其内部标签有返回内容时才会动态语句上插入 WHERE 条件语句。并且, 如果 WHERE 子句以 AND 或者 OR 打头, 则打头的 AND 或 OR 将会被移除。

如果 tutor_id 参数值为 null, 并且 courseName 参数值不为 null, 则<where>标签会将 AND name like #{courseName} 中的 AND 移除掉, 生成的 SQL WHERE 子句为: where name like #{courseName}。

3.6.4 <trim>条件

<trim>元素和<where>元素类似, 但是<trim>提供了在添加前缀/后缀 或者 移除前缀/后缀方面提供更大的灵活性。

XML Code

```
1 <select id="searchCourses" parameterType="hashmap"
2 resultMap="CourseResult">
3     SELECT * FROM COURSES
4     <trim prefix="WHERE" prefixOverrides="AND | OR">
5         <if test=" tutorId != null ">
```

```

6 TUTOR_ID= #{tutorId}
7 </if>
8 <if test="courseName != null">
9 AND name like #{courseName}
10 </if>
11 </trim>
12 </select>

```

这里如果任意一个<if>条件为 true , <trim>元素会插入 WHERE,并且移除紧跟 WHERE 后面的 AND 或 OR

3.6.5 foreach 循环

另外一个强大的动态 SQL 语句构造标签即是<foreach>。它可以迭代遍历一个数组或者列表,构造 AND/OR 条件或一个 IN 子句。

假设我们想找到 tutor_id 为 1, 3, 6 的讲师所教授的课程,我们可以传递一个 tutor_id 组成的列表给映射语句,然后通过<foreach>遍历此列表构造动态 SQL。

XML Code

```

1 <select id="searchCoursesByTutors" parameterType="map"
2 resultMap="CourseResult">
3 SELECT * FROM COURSES
4 <if test="tutorIds != null">
5 <where>
6 <foreach item="tutorId" collection="tutorIds">
7 OR tutor_id=#{tutorId}
8 </foreach>
9 </where>
10 </if>
11 </select>

```

Java Code

```

1 public interface CourseMapper
2 {
3     List<Course> searchCoursesByTutors(Map<String, Object> map);
4 }
5 public void searchCoursesByTutors()
6 {
7     Map<String, Object> map = new HashMap<String, Object>();
8     List<Integer> tutorIds = new ArrayList<Integer>();
9     tutorIds.add(1);
10    tutorIds.add(3);
11    tutorIds.add(6);

```

```

12     map.put("tutorIds", tutorIds);
13     CourseMapper mapper =
14         sqlSession.getMapper(CourseMapper.class);
15     List<Course> courses = mapper.searchCoursesByTutors(map);
16     for (Course course : courses)
17     {
18         System.out.println(course);
19     }
20 }

```

现在让我们来看一下怎样使用<foreach>生成 IN 子句：

XML Code

```

1 <select id="searchCoursesByTutors" parameterType="map"
2 resultMap="CourseResult">
3     SELECT * FROM COURSES
4     <if test="tutorIds != null">
5         <where>
6             tutor_id IN
7                 <foreach item="tutorId" collection="tutorIds"
8                     open="(" separator="," close=")">
9                     #{tutorId}
10                </foreach>
11         </where>
12     </if>
13 </select>

```

3.6.6 set 条件

<set>元素和<where>元素类似，如果其内部条件判断有任何内容返回时，他会插入 SET SQL 片段。

XML Code

```

1 <update id="updateStudent" parameterType="Student">
2     update students
3     <set>
4         <if test="name != null">name=#{name},</if>
5         <if test="email != null">email=#{email},</if>
6         <if test="phone != null">phone=#{phone},</if>
7     </set>
8     where stud_id=#{id}
9 </update>

```


这里，如果<if>条件返回了任何文本内容，<set>将会插入 set 关键字和其文本内容，并且会剔除将末尾的 “,”。

在上述的例子中，如果 phone!=null,<set>将会让会移除 phone=#{phone}后的逗号“,”，生成 set phone=#{phone}。

3.7 MyBaits 食谱

除了简化数据库编程外，MyBatis 还提供了各种功能，这些对实现一些常用任务非常有用，比如按页加载表数据，存取 CLOB/BLOB 类型的数据，处理枚举类型值，等等。让我们来看看其中一些特性吧。

3.7.1 处理枚举类型

MyBatis 支持开箱方式持久化 enum 类型属性。假设 STUDENTS 表中有一列 gender（性别）类型为 varchar，存储“MALE”或者“FEMALE”两种值。并且，Student 对象有一个 enum 类型的 gender 属性，如下所示：

Java Code

```
1 public enum Gender
2 {
3     FEMALE,
4     MALE
5 }
```

默认情况下，MyBatis 使用 EnumTypeHandler 来处理 enum 类型的 Java 属性，并且将其存储为 enum 值的名称。你不需要为此做任何额外的配置。你可以向使用基本数据类型属性一样使用 enum 类型属性，代码如下：

Java Code

```
1 public class Student
2 {
3     private Integer id;
4     private String name;
5     private String email;
6     private PhoneNumber phone;
7     private Address address;
8     private Gender gender;
9     //setters and getters
10 }
```

XML Code

```
1 <insert id="insertStudent" parameterType="Student"
2 useGeneratedKeys="true" keyProperty="id">
3 insert into students(name,email,addr_id, phone,gender)
4 values("#{name}",#{email},#{address.addrId},#{phone},#{gender})
</insert>
```

当你执行 `insertStudent` 语句的时候,MyBatis 会取 Gender 枚举(FEMALE/MALE)的名称,然后将其存储到 GENDER 列中。

如果你希望存储原 enum 的顺序位置,而不是 enum 名,,你需要明确地配置它。

如果你想存储 FEMALE 为 0, MALE 为 1 到 gender 列中,你需要在 `mybatis-config.xml` 文件中配置 `EnumOrdinalTypeHandler`:

XML Code

```
1 <typeHandler
2   handler="org.apache.ibatis.type.EnumOrdinalTypeHandler"
3   javaType="com.mybatis3.domain.Gender"/>
```



使用顺序位置为值存储到数据库时要当心。顺序值是根据 enum 中的声明顺序赋值的。如果你改变了 Gender enum 的声明顺序,则数据库存储的数据和此顺序值就不匹配了。

3.7.2 处理 CLOB/BLOB 类型数据

MyBatis 提供了内建的对 CLOB/BLOB 类型列的映射处理支持。

假设我们有如下的表结构来存储学生和讲师的照片和简介信息:

SQL Code

```
1 CREATE TABLE USER_PICS
2 (
3   ID INT(11) NOT NULL AUTO_INCREMENT,
4   NAME VARCHAR(50) DEFAULT NULL,
5   PIC BLOB,
6   BIO LONGTEXT,
7   PRIMARY KEY (ID)
8 ) ENGINE=INNODB AUTO_INCREMENT=1 DEFAULT CHARSET=LATIN1;
```

这里,照片可以是 PNG, JPG 或其他格式的。简介信息可以是学生或者讲师的漫长的人生经历。默认情况下,MyBatis 将 CLOB 类型的列映射到 `java.lang.String` 类型上、而把 BLOB 列映射到 `byte[]` 类型上。

Java Code

```

1 public class UserPic
2 {
3     private int id;
4     private String name;
5     private byte[] pic;
6     private String bio;
7     //setters & getters
8 }

```

创建 UserPicMapper.xml 文件，配置映射语句，代码如下：

XML Code

```

1 <insert id="insertUserPic" parameterType="UserPic">
2     INSERT INTO USER_PICS(NAME, PIC,BIO)
3     VALUES(#{name},#{pic},#{bio})
4 </insert>
5 <select id="getUserPic" parameterType="int" resultType="UserPic">
6     SELECT * FROM USER_PICS WHERE ID=#{id}
7 </select>

```

下列的 insertUserPic() 展示了如何将数据插入到 CLOB/BLOB 类型的列上：

Java Code

```

1 public void insertUserPic()
2 {
3     byte[] pic = null;
4     try
5     {
6         File file = new File("C:\\Images\\UserImg.jpg");
7         InputStream is = new FileInputStream(file);
8         pic = new byte[is.available()];
9         is.read(pic);
10        is.close();
11    }
12    catch (FileNotFoundException e)
13    {
14        e.printStackTrace();
15    }
16    catch (IOException e)
17    {
18        e.printStackTrace();
19    }
20    String name = "UserName";

```

```

20     String bio = "put some lengthy bio here";
21     UserPic userPic = new UserPic(0, name, pic , bio);
22     SqlSession sqlSession = MyBatisUtil.openSession();
23     try
24     {
25         UserPicMapper mapper =
26             sqlSession.getMapper(UserPicMapper.class);
27         mapper.insertUserPic(userPic);
28         sqlSession.commit();
29     }
30     finally
31     {
32         sqlSession.close();
33     }
34 }
35

```

下面的 getUserPic() 方法展示了怎样将 CLOB 类型数据读取到 String 类型，BLOB 类型数据读取成 byte[] 属性：

Java Code

```

1 public void getUserPic()
2 {
3     UserPic userPic = null;
4     SqlSession sqlSession = MyBatisUtil.openSession();
5     try
6     {
7         UserPicMapper mapper =
8             sqlSession.getMapper(UserPicMapper.class);
9         userPic = mapper.getUserPic(1);
10    }
11    finally
12    {
13        sqlSession.close();
14    }
15    byte[] pic = userPic.getPic();
16    try
17    {
18        OutputStream os = new FileOutputStream(new
19                                                    File("C:\\Images\\UserImage_FromDB.jpg"));
20        os.write(pic);
21        os.close();
22    }
23    catch (FileNotFoundException e)
24    {
25

```

```

24         e.printStackTrace();
25     }
26     catch (IOException e)
27     {
28         e.printStackTrace();
29     }
30 }
31

```

3.7.3 传入多个输入参数

MyBatis 中的映射语句有一个 `parameterType` 属性来制定输入参数的类型。如果我们想给映射语句传入多个参数的话，我们可以将所有的输入参数放到 `HashMap` 中，将 `HashMap` 传递给映射语句。

MyBatis 还提供了另外一种传递多个输入参数给映射语句的方法。假设我们想通过给定的 `name` 和 `email` 信息查找学生信息，定义查询接口如下：

Java Code

```

1 public interface StudentMapper
2 {
3     List<Student> findAllStudentsByNameEmail(String name, String email);
4 }

```

MyBatis 支持 将多个输入参数传递给映射语句，并以 `#{param}` 的语法形式引用它们：

XML Code

```

1 <select id="findAllStudentsByNameEmail" resultMap="StudentResult">
2     select stud_id, name,email, phone from Students
3     where name=#{param1} and email=#{param2}
4 </select>

```

这里 `#{param1}` 引用第一个参数 `name`，而 `#{param2}` 引用了第二个参数 `email`。

Java Code

```

1 StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.
2 class);
3 studentMapper.findAllStudentsByNameEmail(name, email);

```

3.7.4 多行结果集映射成 Map

如果你有一个映射语句返回多行记录，并且你想以 `HashMap` 的形式存储记录的值，使用记录列名作为 `key` 值，而记录对应值或为 `value` 值。我们可以使用 `sqlSession.selectMap()`，如下所示：

XML Code

```
1 <select id="findAllStudents" resultMap="StudentResult">
2     select * from Students
3 </select>
```

Java Code

```
1 Map<Integer, Student> studentMap =
2 sqlSession.selectMap("com.mybatis3.mappers.StudentMapper.findAllStudents", "studId");
```

这里 studentMap 将会将 studId 作为 key 值，而 Student 对象作为 value 值。

3.7.5 使用 RowBounds 对结果集进行分页

有时候，我们会需要跟海量的数据打交道，比如一个有数百万条数据级别的表。由于计算机内存的现实我们不可能一次性加载这么多数据，我们可以获取到数据的一部分。特别是在 Web 应用程序中，分页机制被用来以一页一页的形式展示海量的数据。

MyBatis 可以使用 RowBounds 逐页加载表数据。RowBounds 对象可以使用 offset 和 limit 参数来构建。参数 offset 表示开始位置，而 limit 表示要取的记录的数目。

假设如果你想每页加载并显示 25 条学生的记录，你可以使用如下的代码：

XML Code

```
1 <select id="findAllStudents" resultMap="StudentResult">
2     select * from Students
3 </select>
```

然后，你可以加载如下加载第一页数据（前 25 条）：

Java Code

```
1 int offset = 0, limit = 25;
2 RowBounds rowBounds = new RowBounds(offset, limit);
3 List<Student> = studentMapper.getStudents(rowBounds);
```

若要展示第二页，使用 offset=25, limit=25; 第三页，则为 offset=50, limit=25。

3.7.6 使用 ResultSetHandler 自定义结果集 ResultSet 处理

MyBatis 在将查询结果集映射到 JavaBean 方面提供了很大的选择性。但是，有时候我们会遇到由于特定的目的，需要我们自己处理 SQL 查询结果的情况。MyBatis 提供了 ResultSetHandler 插件形式允许我们以任何自己喜欢的方式处理结果集 ResultSet。

假设我们想从学生的 stud_id 被用作 key，而 name 被用作 value 的 HashMap 中获取到 student 信息。



mybatis-3.2.2 并不支持使用 resultMap 配置将查询的结果集映射成一个属性为 key，而另外属性为 value 的 HashMap。

sqlSession.selectMap()则可以返回 以给定列为 key，记录对象为 value 的 map。我们不能将其配置成使用其中一个属性作为 key，而另外的属性作为 value。

对于 sqlSession.select()方法，我们可以传递给它一个 ResultHandler 的实现，它会被调用来处理 ResultSet 的每一条记录。

Java Code

```
1 public interface ResultHandler
2 {
3     void handleResult(ResultContext context);
4 }
```

现在然我们来看一下怎么使用 ResultHandler 来处理结果集 ResultSet，并返回自定义化的结果。

Java Code

```
1 public Map<Integer, String> getStudentIdNameMap()
2 {
3     final Map<Integer, String> map = new HashMap<Integer, String>();
4     SqlSession sqlSession = MyBatisUtil.openSession();
5     try
6     {
7         sqlSession.select("com.mybatis3.mappers.StudentMapper.findAllStude
8                             nts",
9                             new ResultHandler()
10                            {
11                                @Override
12                                public void handleResult(ResultContext context)
13                                {
14                                    Student student = (Student) context.getResultObject();
15                                    map.put(student.getStudId(), student.getName());
16                                }
17                            }
18                );
19     }
20     finally
21     {
22         sqlSession.close();
```

```

23     }
24     return map;
25 }

```

在上述的代码中，我们提供了匿名内部 `ResultHandler` 实现类，在 `handleResult()` 方法中，我们使用 `context.getResultObject()` 获取当前的 `result` 对象，即 `Student` 对象，因为我们定义了 `findAllStudent` 映射语句的 `resultMap="studentResult"`。对查询返回的每一行都会调用 `handleResult()` 方法，并且我们从 `Student` 对象中取出 `studId` 和 `name`，将其放到 `map` 中。

3.7.7 缓存

将从数据库加载的数据缓存到内存中，是很多应用程序为了提高性能而采取的一贯做法。MyBatis 对通过映射的 `SELECT` 语句加载的查询结果提供了内建的缓存支持。默认情况下，启用一级缓存；即，如果你使用同一个 `SqlSession` 接口对象调用了相同的 `SELECT` 语句，则直接会从缓存中返回结果，而不是再查询一次数据库。

我们可以在 SQL 映射器 XML 配置文件中使用 `<cache />` 元素添加全局二级缓存。

当你加入了 `<cache />` 元素，将会出现以下情况：

- 所有的在映射语句文件定义的 `<select>` 语句的查询结果都会被缓存
- 所有的在映射语句文件定义的 `<insert>`, `<update>` 和 `<delete>` 语句将会刷新缓存
- 缓存根据最近最少被使用 (Least Recently Used, LRU) 算法管理
- 缓存不会被任何形式的基于时间表的刷新 (没有刷新时间间隔)，即不支持定时刷新机制
- 缓存将存储 1024 个 查询方法返回的列表或者对象的引用
- 缓存会被当作一个读/写缓存。这指的是检索出的对象不会被共享，并且可以被调用者安全地修改，不会其他潜在的调用者或者线程的潜在修改干扰。(即，缓存是线程安全的)

你也可以通过复写默认属性来自定义缓存的行为，如下所示：

XML Code

```

1 <cache eviction="FIFO" flushInterval="60000" size="512"
2   readOnly="true"/>

```

以下是对上述属性的描述：

- `eviction`: 此处定义缓存的移除机制。默认值是 LRU (least recently used, 最近最少使用), FIFO (first in first out, 先进先出), SOFT (soft reference, 软引用), WEAK (weak reference, 弱引用)。
- `flushInterval`: 定义缓存刷新间隔，以毫秒计。默认情况下不设置。所以不使用刷新间隔，缓存 `cache` 只有调用语句的时候刷新。
- `size`: 此表示缓存 `cache` 中能容纳的最大元素数。默认值是 1024，你可以设置成任意的正整数。
- `readOnly`: 一个只读的缓存 `cache` 会对所有的调用者返回被缓存对象的同一个实例 (实际返回的是被返回对象的一份引用)。一个读/写缓存 `cache` 将会返回被返回对象的一份拷贝 (通过序列化)。默认情况下设置为 `false`。可能的值有 `false` 和 `true`。

一个缓存的配置和缓存实例被绑定到映射器配置文件所在的名空间 (namespace) 上，所以在相同名空间内的所有语句被绑定到一个 `cache` 中。

默认的映射语句的 cache 配置如下：

XML Code

```
1 <select ... flushCache="false" useCache="true"/>
2 <insert ... flushCache="true"/>
3 <update ... flushCache="true"/>
4 <delete ... flushCache="true"/>
```

你可以为任意特定的映射语句复写默认的 cache 行为；例如，对一个 select 语句不使用缓存，可以设置 useCache="false"。

除了内建的缓存支持，MyBatis 也提供了与第三方缓存类库如 Ehcache，OSCache，Hazelcast 的集成支持。你可以在 MyBatis 官方网站 <https://code.google.com/p/mybatis/wiki/Caches> 上找到关于继承第三方缓存类库的更多信息。

3.8 总结

在本章中，我们学习了怎样使用映射器配置文件 书写 SQL 映射语句。讨论了如何配置简单的语句，一对一以及一对多关系的语句，以及怎样使用 ResultMap 进行结果集映射。我们还了解了构建动态 SQL 语句，结果分页，以及自定义结果集（ResultSet）处理。在下一章，我们将会讨论如何使用注解书写映射语句。

4

第四章 使用注解配置 SQL 映射器

在上一章，我们看到了我们是怎样在映射器 Mapper XML 配置文件中配置映射语句的。MyBatis 也支持使用注解来配置映射语句。当我们使用基于注解的映射器接口时，我们不再需要在 XML 配置文件中配置了。如果你愿意，你也可以同时使用基于 XML 和基于注解的映射语句。

本章将涵盖以下话题：

- 在映射器 Mapper 接口上使用注解
- 映射语句
 - @Insert, @Update, @Delete, @SelectStatements
- 结果映射
 - 一对一映射
 - 一对多映射
- 动态 SQL
 - @SelectProvider
 - @InsertProvider
 - @UpdateProvider
 - @DeleteProvider

4.1 在映射器 Mapper 接口上使用注解

MyBatis 对于大部分的基于 XML 的映射器元素 (包括<select>,<update>) 提供了对应的基于注解的配置项。然而在某些情况下, 基于注解配置 还不能支持基于 XML 的一些元素。

4.2 映射语句

MyBatis 提供了多种注解来支持不同类型的语句(statement)如 SELECT,INSERT,UPDATE,DELETE。让我们看一下具体怎样配置映射语句。

4.2.1 @Insert

我们可以使用@Insert 注解来定义一个 INSERT 映射语句：

Java Code

```
1 package com.mybatis3.mappers;
2 public interface StudentMapper
3 {
4     @Insert("INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL,ADDR_ID, PHONE)
5             VALUES(#{studId},#{name},#{email},#{address.addrId},#{phone})")
6     int insertStudent(Student student);
7 }
```

使用了@Insert 注解的 insertMethod()方法将返回 insert 语句执行后影响的行数。

[自动生成主键]

在上一章中我们讨论过主键列值可以自动生成。我们可以使用@Options 注解的 userGeneratedKeys 和 keyProperty 属性让数据库产生 auto_increment (自增长) 列的值, 然后将生成的值设置到输入参数对象的属性中。

Java Code

```
1 @Insert("INSERT INTO STUDENTS(NAME,EMAIL,ADDR_ID, PHONE)
2         VALUES(#{name},#{email},#{address.addrId},#{phone})")
3 @Options(useGeneratedKeys = true, keyProperty = "studId")
4 int insertStudent(Student student);
```

这里 STUD_ID 列值将会通过 MySQL 数据库自动生成。并且生成的值将会被设置到 student 对象的 studId 属性中。

Java Code

```
1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 mapper.insertStudent(student);
3 int studId = student.getStudId();
```

有一些数据库如 Oracle, 并不支持 AUTO_INCREMENT 列属性, 它使用序列 (SEQUENCE) 来产生主键的值。

我们可以使用@SelectKey 注解来为任意 SQL 语句来指定主键值，作为主键列的值。
假设我们有一个名为 STUD_ID_SEQ 的序列来生成 STUD_ID 主键值。

Java Code

```
1 @Insert("INSERT INTO STUDENTS(STUD_ID,NAME,EMAIL,ADDR_ID, PHONE)
2 VALUES(#{studId},#{name},#{email},#{address.addrId},#{phone})")
3 @SelectKey(statement="SELECT STUD_ID_SEQ.NEXTVAL FROM DUAL",
4 keyProperty="studId", resultType=int.class, before=true)
5 int insertStudent(Student student);
```

这里我们使用了@SelectKey 来生成主键值，并且存储到了 student 对象的 studId 属性上。由于我们设置了 before=true,该语句将会在执行 INSERT 语句之前执行。

如果你使用序列作为触发器来设置主键值，我们可以在 INSERT 语句执行后，从 sequence_name.currval 获取数据库产生的主键值。

Java Code

```
1 @Insert("INSERT INTO STUDENTS(NAME,EMAIL,ADDR_ID, PHONE)
2 VALUES(#{name},#{email},#{address.addrId},#{phone})")
3 @SelectKey(statement="SELECT STUD_ID_SEQ.CURRVAL FROM DUAL",
4 keyProperty="studId", resultType=int.class, before=false)
5 int insertStudent(Student student);
```

4.2.2 @Update

我们可以使用@Update 注解来定义一个 UPDATE 映射语句，如下所示：

Java Code

```
1 @Update("UPDATE STUDENTS SET NAME=#{name}, EMAIL=#{email},
2 PHONE=#{phone} WHERE STUD_ID=#{studId}")
3 int updateStudent(Student student);
```

使用了@Update 的 updateStudent()方法将会返回执行了 update 语句后影响的行数。

Java Code

```
1 StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
2 int noOfRowsUpdated = mapper.updateStudent(student);
```

4.2.3 @Delete

我们可以使用@Delete 注解来定义一个 DELETE 映射语句，如下所示：

Java Code

```

1 @Delete("DELETE FROM STUDENTS WHERE STUD_ID=#{studId}")
2 int deleteStudent(int studId);

```

使用了@Delete 的 deleteStudent()方法将会返回执行了 update 语句后影响的行数。

4.2.4 @Select

我们可以使用@ Select 注解来定义一个 SELECT 映射语句。

让我们看一下怎样使用注解配置一个简单的 select 查询。

Java Code

```

1 package com.mybatis3.mappers;
2 public interface StudentMapper
3 {
4     @Select("SELECT STUD_ID AS STUDID, NAME, EMAIL, PHONE FROM
5             STUDENTS WHERE STUD_ID=#{studId}")
6     Student findStudentById(Integer studId);
7 }

```

为了将列名和 Student bean 属性名匹配，我们为 stud_id 起了一个 studId 的别名。如果返回了多行结果，将抛出 TooManyResultsException 异常。

4.3 结果映射

我们可以将查询结果通过别名或者是@Results 注解与 JavaBean 属性映射起来。

现在让我们看看怎样使用@Results 注解将指定列于指定 JavaBean 属性映射器来，执行 SELECT 查询的：

Java Code

```

1 package com.mybatis3.mappers;
2 public interface StudentMapper
3 {
4     @Select("SELECT * FROM STUDENTS")
5     @Results(
6     {
7         @Result(id = true, column = "stud_id", property = "studId"),
8         @Result(column = "name", property = "name"),
9         @Result(column = "email", property = "email"),
10        @Result(column = "addr_id", property = "address.addrId")
11    })
12    List<Student> findAllStudents();
13 }

```



`@Results` 注解和映射器 XML 配置文件元素 `<resultMap>` 想对应。然而，MyBatis3.2.2 不能为 `@Results` 注解赋予一个 ID。所以，不像 `<resultMap>` 元素，我们不应在不同的映射语句中重用 `@Results` 声明。这意味着即使 `@Results` 注解完全相同，我们也需要(在不同的映射接口中)重复 `@Results` 声明。

例如，看下面的 `findStudentById()` 和 `findAllStudents()` 方法：

Java Code

```
1 @Select("SELECT * FROM STUDENTS WHERE STUD_ID=#{studId}")
2 @Results(
3 {
4     @Result(id = true, column = "stud_id", property = "studId"),
5     @Result(column = "name", property = "name"),
6     @Result(column = "email", property = "email"),
7     @Result(column = "addr_id", property = "address.addrId")
8 })
9 Student findStudentById(int studId);
10 @Select("SELECT * FROM STUDENTS")
11 @Results(
12 {
13     @Result(id = true, column = "stud_id", property = "studId"),
14     @Result(column = "name", property = "name"),
15     @Result(column = "email", property = "email"),
16     @Result(column = "addr_id", property = "address.addrId")
17 })
18 List<Student> findAllStudents();
```

这里两个语句的 `@Results` 配置完全相同，但是我必须得重复它。这里有一个解决方法。我们可以创建一个映射器 Mapper 配置文件，然后配置 `<resultMap>` 元素，然后使用 `@ResultMap` 注解引用此 `<resultMap>`。

在 `StudentMapper.xml` 中定义一个 ID 为 `StudentResult` 的 `<resultMap>`。

XML Code

```
1 <mapper namespace="com.mybatis3.mappers.StudentMapper">
2   <resultMap type="Student" id="StudentResult">
3     <id property="studId" column="stud_id" />
4     <result property="name" column="name" />
5     <result property="email" column="email" />
6     <result property="phone" column="phone" />
7   </resultMap>
8 </mapper>
9
```

在 `StudentMapper.java` 中，使用 `@ResultMap` 引用名为 `StudentResult` 的 `resultMap`。

Java Code

```
1 public interface StudentMapper
2 {
3     @Select("SELECT * FROM STUDENTS WHERE STUD_ID=#{studId}")
4     @ResultMap("com.mybatis3.mappers.StudentMapper.StudentResult")
5     Student findStudentById(int studId);
6     @Select("SELECT * FROM STUDENTS")
7     @ResultMap("com.mybatis3.mappers.StudentMapper.StudentResult")
8     List<Student> findAllStudents();
9 }
```

4.3.1 一对一映射

MyBatis 提供了@One 注解来使用嵌套 select 语句 (Nested-Select) 加载一对一关联查询数据。让我们看看怎样使用@One 注解获取学生及其地址信息。

Java Code

```
1 public interface StudentMapper
2 {
3     @Select("SELECT ADDR_ID AS ADDRID, STREET, CITY, STATE, ZIP, COUNTRY
4             FROM ADDRESSES WHERE ADDR_ID=#{id}")
5     Address findAddressById(int id);
6     @Select("SELECT * FROM STUDENTS WHERE STUD_ID=#{studId} ")
7     @Results(
8     {
9         @Result(id = true, column = "stud_id", property = "studId"),
10        @Result(column = "name", property = "name"),
11        @Result(column = "email", property = "email"),
12        @Result(property = "address", column = "addr_id",
13        one = @One(select = "com.mybatis3.mappers.StudentMapper.
14        findAddressById"))
15    })
16    Student selectStudentWithAddress(int studId);
17 }
```

这里我们使用了@One 注解的 select 属性来指定一个使用了完全限定名的方法上，该方法会返回一个 Address 对象。使用 column="addr_id", 则 STUDENTS 表中列 addr_id 的值将会作为输入参数传递给 findAddressById() 方法。如果@One SELECT 查询返回了多行结果，则会抛出 TooManyResultsException 异常。

Java Code

```
1 int studId = 1;
2 StudentMapper studentMapper =
3 sqlSession.getMapper(StudentMapper.class);
4 Student student = studentMapper.selectStudentWithAddress(studId);
```

```

4 System.out.println("Student :"+student);
5 System.out.println("Address :"+student.getAddress());
6

```

在第三章,使用 XML 配置 SQL 映射器中我们讨论过,我们可以通过基于 XML 的映射器配置,使用嵌套结果 resultMap 来加载一对关联的查询。而 MyBatis3.2.2 版本,并没有对应的注解支持。但是我们可以在映射器 Mapper 配置文件中配置<resultMap>并且使用@ResultMap 注解来引用它。

在 StudentMapper.xml 中配置<resultMap>, 如下所示:

XML Code

```

1 <mapper namespace="com.mybatis3.mappers.StudentMapper">
2   <resultMap type="Address" id="AddressResult">
3     <id property="addrId" column="addr_id" />
4     <result property="street" column="street" />
5     <result property="city" column="city" />
6     <result property="state" column="state" />
7     <result property="zip" column="zip" />
8     <result property="country" column="country" />
9   </resultMap>
10  <resultMap type="Student" id="StudentWithAddressResult">
11    <id property="studId" column="stud_id" />
12    <result property="name" column="name" />
13    <result property="email" column="email" />
14    <association property="address" resultMap="AddressResult" />
15  </resultMap>
16 </mapper>
17

```

Java Code

```

1 public interface StudentMapper
2 {
3   @Select("select stud_id, name, email, a.addr_id, street, city,
4           state, zip, country" + " FROM students s left outer join addresses a
5           on s.addr_id=a.addr_id" + " where stud_id=#{studId} ")
6   @ResultMap("com.mybatis3.mappers.StudentMapper.
7             StudentWithAddressResult")
8   Student selectStudentWithAddress(int id);
9 }

```

4.3.2 一对多映射

MyBatis 提供了@Many 注解，用来使用嵌套 Select 语句加载一对多关联查询。

现在让我们看一下如何使用@Many 注解获取一个讲师及其教授课程列表信息：

Java Code

```
1 public interface TutorMapper
2 {
3     @Select("select addr_id as addrId, street, city, state, zip,
4             country from addresses where addr_id=#{id}")
5     Address findAddressById(int id);
6     @Select("select * from courses where tutor_id=#{tutorId}")
7     @Results(
8     {
9         @Result(id = true, column = "course_id", property = "courseId"),
10        @Result(column = "name", property = "name"),
11        @Result(column = "description", property = "description"),
12        @Result(column = "start_date" property = "startDate"),
13        @Result(column = "end_date" property = "endDate")
14    })
15    List<Course> findCoursesByTutorId(int tutorId);
16    @Select("SELECT tutor_id, name as tutor_name, email, addr_id
17            FROM tutors where tutor_id=#{tutorId}")
18    @Results(
19    {
20        @Result(id = true, column = "tutor_id", property = "tutorId"),
21        @Result(column = "tutor_name", property = "name"),
22        @Result(column = "email", property = "email"),
23        @Result(property = "address", column = "addr_id",
24        one = @One(select = " com.mybatis3.
25        mappers.TutorMapper.findAddressById")),
26        @Result(property = "courses", column = "tutor_id",
27        many = @Many(select = "com.mybatis3.mappers.TutorMapper.
28        findCoursesByTutorId"))
29    })
30    Tutor findTutorById(int tutorId);
31 }
```

这里我们使用了@Many 注解的 select 属性来指向一个完全限定名称的方法，该方法将返回一个 List<Course>对象。使用 column="tutor_id"，TUTORS 表中的 tutor_id 列值将会作为输入参数传递给 findCoursesByTutorId()方法。

在第三章，使用 XML 配置 SQL 映射器中我们讨论过，我们可以通过基于 XML 的映射器配置，使用嵌套结果 resultMap 来加载一对多关联的查询。而 MyBatis3.2.2 版本，并没有对应的注解支持。但是我们可以在映射器 Mapper 配置文件中配置<resultMap>并且使用@ResultMap 注解来引用它。

在 TutorMapper.xml 中配置<resultMap>,如下所示：

XML Code

```
1 <mapper namespace="com.mybatis3.mappers.TutorMapper">
2   <resultMap type="Address" id="AddressResult">
3     <id property="addrId" column="addr_id" />
4     <result property="street" column="street" />
5     <result property="city" column="city" />
6     <result property="state" column="state" />
7     <result property="zip" column="zip" />
8     <result property="country" column="country" />
9   </resultMap>
10  <resultMap type="Course" id="CourseResult">
11    <id column="course_id" property="courseId" />
12    <result column="name" property="name" />
13    <result column="description" property="description" />
14    <result column="start_date" property="startDate" />
15    <result column="end_date" property="endDate" />
16  </resultMap>
17  <resultMap type="Tutor" id="TutorResult">
18    <id column="tutor_id" property="tutorId" />
19    <result column="tutor_name" property="name" />
20    <result column="email" property="email" />
21    <association property="address" resultMap="AddressResult" />
22    <collection property="courses" resultMap="CourseResult" />
23  </resultMap>
24 </mapper>
25
```

Java Code

```
1 public interface TutorMapper
2 {
3   @Select("SELECT T.TUTOR_ID, T.NAME AS TUTOR_NAME, EMAIL,
4           A.ADDR_ID, STREET, CITY, STATE, ZIP, COUNTRY, COURSE_ID, C.NAME,
5           DESCRIPTION, START_DATE, END_DATE FROM TUTORS T LEFT OUTER
6           JOIN ADDRESSES A ON T.ADDR_ID=A.ADDR_ID LEFT OUTER JOIN COURSES
7           C ON T.TUTOR_ID=C.TUTOR_ID WHERE T.TUTOR_ID=#{tutorId}")
8   @ResultMap("com.mybatis3.mappers.TutorMapper.TutorResult")
9   Tutor selectTutorById(int tutorId);
10 }
```

4.4 动态 SQL

有时候我们需要根据输入条件动态地构建 SQL 语句。MyBatis 提供了各种注解如 `@InsertProvider`, `@UpdateProvider`, `@DeleteProvider` 和 `@SelectProvider`，来帮助构建动态 SQL 语句，然后让 MyBatis 执行这些 SQL 语句。

4.4.1 @SelectProvider

现在让我们来看一个使用 `@SelectProvider` 注解来创建一个简单的 SELECT 映射语句的例子。

创建一个 `TutorDynaSqlProvider.java` 类，以及 `findTutorByIdSql()` 方法，如下所示：

Java Code

```
1 package com.mybatis3.sqlproviders;
2 import org.apache.ibatis.jdbc.SQL;
3 public class TutorDynaSqlProvider
4 {
5     public String findTutorByIdSql(int tutorId)
6     {
7         return "SELECT TUTOR_ID AS tutorId, NAME, EMAIL FROM TUTORS
8             WHERE TUTOR_ID=" + tutorId;
9     }
10 }
```

在 `TutorMapper.java` 接口中创建一个映射语句，如下：

Java Code

```
1 @SelectProvider(type=TutorDynaSqlProvider.class, method="findTutorByIdSql")
2 Tutor findTutorById(int tutorId);
```

这里我们使用了 `@SelectProvider` 来指定了一个类，及其内部的方法，用来提供需要执行的 SQL 语句。

但是使用字符串拼接的方法构建 SQL 语句是非常困难的，并且容易出错。所以 MyBatis 提供了一个 SQL 工具类不使用字符串拼接的方式，简化构造动态 SQL 语句。

现在，让我们看看如何使用 `org.apache.ibatis.jdbc.SQL` 工具类来准备相同的 SQL 语句。

Java Code

```
1 package com.mybatis3.sqlproviders;
2 import org.apache.ibatis.jdbc.SQL;
3 public class TutorDynaSqlProvider
4 {
5     public String findTutorByIdSql(final int tutorId)
6     {
7         return new SQL()
8         {
9             {
10                 SELECT("tutor_id as tutorId, name, email");
11                 FROM("tutors");
12             }
13         }
14     }
15 }
```

```

11         WHERE("tutor_id=" + tutorId);
12     }
13     } .toString();
14 }
15 }
16

```

SQL 工具类会处理以合适的空格前缀和后缀来构造 SQL 语句。

动态 SQL **provider** 方法可以接收以下其中一种参数：

- 无参数
- 和映射器 Mapper 接口的方法同类型的参数
- java.util.Map

如果 SQL 语句的准备不取决于输入参数，你可以使用不带参数的 SQL Provider 方法。

例如：

Java Code

```

1 public String findTutorByIdSql()
2 {
3     return new SQL()
4     {
5         {
6             SELECT("tutor_id as tutorId, name, email");
7             FROM("tutors");
8             WHERE("tutor_id = #{tutorId}");
9         }
10    } .toString();
11 }

```

这里我们没有使用输入参数构造 SQL 语句，所以它可以是一个无参方法。

如果映射器 Mapper 接口方法只有一个参数，那么可以定义 SQL Provider 方法，它接受一个与 Mapper 接口方法相同类型的参数。

例如映射器 Mapper 接口有如下定义：

Java Code

```

1 Tutor findTutorById(int tutorId);

```

这里 findTutorById(int)方法只有一个 int 类型的参数。我们可以定义 findTutorByIdSql(int)方法作为 SQL provider 方法。

Java Code

```

1 public String findTutorByIdSql(final int tutorId)
2 {

```

```

3     return new SQL()
4     {
5         {
6             SELECT("tutor_id as tutorId, name, email");
7             FROM("tutors");
8             WHERE("tutor_id=" + tutorId);
9         }
10    } .toString();
11 }

```

如果映射器 Mapper 接口有多个输入参数，我们可以使用参数类型为 `java.util.Map` 的方法作为 SQL provider 方法。然后映射器 Mapper 接口方法所有的输入参数将会被放到 map 中，以 `param1`, `param2` 等等作为 key，将输入参数按序作为 value。你也可以使用 `0`, `1`, `2` 等等作为 key 值来取的输入参数。

Java Code

```

1 @SelectProvider(type = TutorDynaSqlProvider.class,
2                 method = "findTutorByNameAndEmailSql")
3 Tutor findTutorByNameAndEmail(String name, String email);
4 public String findTutorByNameAndEmailSql(Map<String, Object> map)
5 {
6     String name = (String) map.get("param1");
7     String email = (String) map.get("param2");
8     //you can also get those values using 0,1 keys
9     //String name = (String) map.get("0");
10    //String email = (String) map.get("1");
11    return new SQL()
12    {
13        {
14            SELECT("tutor_id as tutorId, name, email");
15            FROM("tutors");
16            WHERE("name=#{name} AND email=#{email}");
17        }
18    } .toString();
19 }

```

SQL 工具类也提供了其他的方法来表示 JOINS，ORDER_BY，GROUP_BY 等等。
让我们看一个使用 LEFT_OUTER_JOIN 的例子：

Java Code

```

1 public class TutorDynaSqlProvider
2 {
3     public String selectTutorById()
4     {

```

```

5         return new SQL()
6         {
7             {
8                 SELECT("t.tutor_id, t.name as tutor_name, email");
9                 SELECT("a.addr_id, street, city, state, zip, country");
10                SELECT("course_id, c.name as course_name, description,
11                    start_date, end_date");
12                FROM("TUTORS t");
13                LEFT_OUTER_JOIN("addresses a on t.addr_id=a.addr_id");
14                LEFT_OUTER_JOIN("courses c on t.tutor_id=c.tutor_id");
15                WHERE("t.TUTOR_ID = #{id}");
16            }
17        } .toString();
18    }
19 }
20
21 public interface TutorMapper
22 {
23     @SelectProvider(type = TutorDynaSqlProvider.class,
24         method = "selectTutorById")
25     @ResultMap("com.mybatis3.mappers.TutorMapper.TutorResult")
26     Tutor selectTutorById(int tutorId);
27 }

```

由于没有支持使用内嵌结果 resultMap 的一对多关联映射的注解支持，我们可以使用基于 XML 的<resultMap>配置，然后与@ResultMap 映射。

XML Code

```

1 <mapper namespace="com.mybatis3.mappers.TutorMapper">
2     <resultMap type="Address" id="AddressResult">
3         <id property="id" column="addr_id" />
4         <result property="street" column="street" />
5         <result property="city" column="city" />
6         <result property="state" column="state" />
7         <result property="zip" column="zip" />
8         <result property="country" column="country" />
9     </resultMap>
10    <resultMap type="Course" id="CourseResult">
11        <id column="course_id" property="id" />
12        <result column="course_name" property="name" />
13        <result column="description" property="description" />
14        <result column="start_date" property="startDate" />
15        <result column="end_date" property="endDate" />
16    </resultMap>
17    <resultMap type="Tutor" id="TutorResult">

```

```

17     <id column="tutor_id" property="id" />
18     <result column="tutor_name" property="name" />
19     <result column="email" property="email" />
20     <association property="address" resultMap="AddressResult" />
21     <collection property="courses" resultMap="CourseResult"></collection>
22 </resultMap>
23 </mapper>
24
25

```

使用了动态的 SQL provider，我们可以取得讲师及其地址和课程明细。

4.4.2 @InsertProvider

我们可以使用@InsertProvider 注解创建动态的 INSERT 语句，如下所示：

Java Code

```

1 public class TutorDynaSqlProvider
2 {
3     public String insertTutor(final Tutor tutor)
4     {
5         return new SQL()
6         {
7             {
8                 INSERT_INTO("TUTORS");
9                 if (tutor.getName() != null)
10                 {
11                     VALUES("NAME", "#{name}");
12                 }
13                 if (tutor.getEmail() != null)
14                 {
15                     VALUES("EMAIL", "#{email}");
16                 }
17             }
18             } .toString();
19     }
20 }
21 public interface TutorMapper
22 {
23     @InsertProvider(type = TutorDynaSqlProvider.class,
24                     method = "insertTutor")
25     @Options(useGeneratedKeys = true, keyProperty = "tutorId")
26     int insertTutor(Tutor tutor);
27 }

```

4.4.3 @UpdateProvider

我们可以通过@UpdateProvider 注解创建 UPDATE 语句，如下所示：

Java Code

```
1 public class TutorDynaSqlProvider
2 {
3     public String updateTutor(final Tutor tutor)
4     {
5         return new SQL()
6         {
7             {
8                 UPDATE("TUTORS");
9                 if (tutor.getName() != null)
10                {
11                    SET("NAME = #{name}");
12                }
13                if (tutor.getEmail() != null)
14                {
15                    SET("EMAIL = #{email}");
16                }
17                WHERE("TUTOR_ID = #{tutorId}");
18            }
19        }.toString();
20    }
21 }
22 public interface TutorMapper
23 {
24     @UpdateProvider(type = TutorDynaSqlProvider.class,
25                    method = "updateTutor")
26     int updateTutor(Tutor tutor);
27 }
```

4.4.4 @DeleteProvider

我们可以使用@DeleteProvider 注解创建动态地 DELETE 语句，如下所示：

Java Code

```
1 public class TutorDynaSqlProvider
2 {
3     public String deleteTutor(int tutorId)
4     {
5         return new SQL()
6         {
```



```
7         {
8             DELETE_FROM("TUTORS");
9             WHERE("TUTOR_ID = #{tutorId}");
10        }
11    } .toString();
12 }
13 }
14 public interface TutorMapper
15 {
16     @DeleteProvider(type = TutorDynaSqlProvider.class,
17                     method = "deleteTutor")
18     int deleteTutor(int tutorId);
19 }
```

4.5 总结

在本章中，我们学习了怎样使用注解书写 SQL 映射语句。讨论了如何配置简单语句，一对一关系语句和一对多关系语句。我们还探讨了怎样使用 SqlProvider 注解来构建动态 SQL 语句。在下一章 我们将讨论如何将 MyBatis 与 Spring 框架集成。

第五章 与 Spring 集成

MyBatis-Spring 是 MyBatis 框架的子模块，用来提供与当前流行的依赖注入框架 Spring 的无缝集成。

Spring 框架是一个基于依赖注入 (Dependency Injection) 和面向切面编程 (Aspect Oriented Programming, AOP) 的 Java 框架，鼓励使用基于 POJO 的编程模型。另外，Spring 提供了声明式和编程式的事务管理能力，可以很大程度上简化应用程序的数据访问层 (data access layer) 的实现。在本章中，我们将看到在基于 Spring 的应用程序中使用 MyBatis 并且使用 Spring 的基于注解的事务管理机制。

本章将包含以下话题：

- 在 Spring 应用程序中配置 MyBatis
 - 安装
 - 配置 MyBatis Beans
- 使用 SqlSession
- 使用映射器
- 使用 Spring 进行事务管理

5.1 在 Spring 应用程序中配置 MyBatis

本节将讨论如何在基于 Spring 的应用程序中安装和配置 MyBatis

5.1.1 安装

如果你正在使用 Maven 构建工具，你可以配置 MyBatis 的 spring 依赖如下：

XML Code

```
1 <dependency>
2   <groupId>org.mybatis</groupId>
3   <artifactId>mybatis-spring</artifactId>
4   <version>1.2.0</version>
5 </dependency>
6 <dependency>
7   <groupId>org.springframework</groupId>
8   <artifactId>spring-context-support</artifactId>
9   <version>3.1.3.RELEASE</version>
10 <exclusions>
```

```

11     <exclusion>
12         <groupId>commons-logging</groupId>
13         <artifactId>commons-logging</artifactId>
14     </exclusion>
15 </exclusions>
16 </dependency>
17 <dependency>
18     <groupId>org.springframework</groupId>
19     <artifactId>spring-jdbc</artifactId>
20     <version>3.1.3.RELEASE</version>
21 </dependency>
22 <dependency>
23     <groupId>org.springframework</groupId>
24     <artifactId>spring-test</artifactId>
25     <version>3.1.3.RELEASE</version>
26     <scope>test</scope>
27 </dependency>
28 <dependency>
29     <groupId>org.aspectj</groupId>
30     <artifactId>aspectjrt</artifactId>
31     <version>1.6.8</version>
32 </dependency>
33 <dependency>
34     <groupId>org.aspectj</groupId>
35     <artifactId>aspectjweaver</artifactId>
36     <version>1.6.8</version>
37 </dependency>
38 <dependency>
39     <groupId>cglib</groupId>
40     <artifactId>cglib-nodep</artifactId>
41     <version>2.2</version>
42 </dependency>
43 <dependency>
44     <groupId>commons-dbcp</groupId>
45     <artifactId>commons-dbcp</artifactId>
46     <version>1.4</version>
47 </dependency>
48

```

如果你没有使用 Maven，你可以从 <http://code.google.com/p/mybatis/> 上下载 mybatis-spring-1.2.0-bundle.zip。将其加入，将 mybatis-1.2.0.jar 包添加到 classpath 中。

你可以从 <http://www.springsource.org/download/community/> 上下载 Spring 框架包 spring-framework-3.1.3.RELEASE.zip。将其内所有 jar 包添加到 classpath 中。

如果你只使用 MyBatis 而没有使用 Spring，在每一个方法中，我们需要手动创建 SqlSessionFactory 对象，并且从 SqlSessionFactory 对象中创建 SqlSession。而且我们还要负责提交或者回滚事务、关闭 SqlSession 对象。

通过使用 MyBatis-Spring 模块，我们可以在 Spring 的应用上下文 ApplicationContext 中配置 MyBatis Beans，Spring 会负责实例化 SqlSessionFactory 对象以及创建 SqlSession 对象，并将其注入到 DAO 或者 Service 类中。并且，你可以使用 Spring 的基于注解的事务管理功能，不用自己在数据访问层中书写事务处理代码了。

5.1.2 配置 MyBatis Beans

为了让 Spring 来实例化 MyBatis 组件如 SqlSessionFactory、SqlSession、以及映射器 Mapper 对象，我们需要在 Spring 的 bean 配置文件中配置它们，假设在 applicationContext.xml 中，配置如下：

XML Code

```
1 <beans>
2   <bean id="dataSource" class="org.springframework.jdbc.datasource. DriverManagerDataSource">
3     <property name="driverClassName" value="com.mysql.jdbc.Driver" />
4     <property name="url" value="jdbc:mysql://localhost:3306/elearning" />
5     <property name="username" value="root" />
6     <property name="password" value="admin" />
7   </bean>
8   <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
9     <property name="dataSource" ref="dataSource" />
10    <property name="typeAliases" value="com.mybatis3.domain.Student, com.mybatis3.domain.Tutor" />
11    <property name="typeAliasesPackage" value="com.mybatis3.domain" />
12    <property name="typeHandlers" value="com.mybatis3.typehandlers.PhoneTypeHandler" />
13    <property name="typeHandlersPackage" value="com.mybatis3.typehandlers" />
14    <property name="mapperLocations" value="classpath*:com/mybatis3/**/*.xml" />
15    <property name="configLocation" value="WEB-INF/mybatisconfig.xml" />
16  </bean>
17 </beans>
```

使用上述的 bean 定义，Spring 会使用如下配置属性创建一个 SqlSessionFactory 对象：

- **dataSource**: 它引用了 dataSource bean
- **typeAliases**: 它指定了一系列的完全限定名的类名列表，用逗号隔开，这些别名将通过默认的别名规则创建（将首字母小写的非无完全限定类名作为别名）。
- **typeAliasesPackage**: 它指定了一系列包名列表，用逗号隔开，包内含有需要创建别名的 JavaBeans。
- **typeHandlers**: 它指定了一系列的类型处理器类的完全限定名的类名列表，用逗号隔开。
- **typeHandlersPackage**: 它指定了一系列包名列表，用逗号隔开，包内含有需要被注册的类型处理器类。
- **mapperLocations**: 它指定了 SQL 映射器 Mapper XML 配置文件的位置
- **configLocation**: 它指定了 MyBatisSqlSessionFactory 配置文件所在的位置。

5.2 使用 SqlSession

一旦 SqlSessionFactory bean 被配置, 我们需要配置 SqlSessionTemplate bean, SqlSessionTemplate bean 是一个线程安全的 Spring bean, 我们可以从中获取到线程安全的 SqlSession 对象。由于 SqlSessionTemplate 提供线程安全的 SqlSession 对象, 你可以在多个 Spring bean 实体对象中共享 SqlSessionTemplate 对象。从概念上看, SqlSessionTemplate 和 Spring 的 DAO 模块中的 JdbcTemplate 非常相似。

XML Code

```
1 <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
2   <constructor-arg index="0" ref="sqlSessionFactory" />
3 </bean>
4
```

现在我肯可以将 SqlSession bean 实体对象注射到任意的 Spring bean 实体中, 然后使用 SqlSession 对象调用 SQL 映射语句。

Java Code

```
1 public class StudentDaoImpl implements StudentDao
2 {
3     private SqlSession sqlSession;
4     public void setSqlSession(SqlSession session)
5     {
6         this.sqlSession = session;
7     }
8     public void createStudent(Student student)
9     {
10         StudentMapper mapper =
11             this.sqlSession.getMapper(StudentMapper.class);
12         mapper.insertStudent(student);
13     }
14 }
```

如果你正在使用基于 XML 来配置 Spring beans, 你可以将 SqlSession bean 实体对象注射到 StudentDaoImpl bean 实体对象中, 如下:

XML Code

```
1 <bean id="studentDao" class="com.mybatis3.dao.StudentDaoImpl">
2   <property name="sqlSession" ref="sqlSession" />
3 </bean>
```

如果你使用基于注解的方式配置 Spring beans, 你如下将 SqlSession bean 实体对象注入到 StudentDaoImpl bean 实体对象中:

Java Code

```

1 @Repository
2 public class StudentDaoImpl implements StudentDao
3 {
4     private SqlSession sqlSession;
5     @Autowired
6     public void setSqlSession(SqlSession session)
7     {
8         this.sqlSession = session;
9     }
10    public void createStudent(Student student)
11    {
12        StudentMapper mapper =
13            this.sqlSession.getMapper(StudentMapper.class);
14        mapper.insertStudent(student);
15    }
16 }

```

还有另外一种注入 `SqlSession` 对象的方法，即，通过拓展继承 `SqlSessionDaoSupport`。这种方式让我们可以在执行映射语句时，加入任何自定义的逻辑。

Java Code

```

1 public class StudentMapperImpl extends SqlSessionDaoSupport implements
2     StudentMapper
3 {
4     public void createStudent(Student student)
5     {
6         StudentMapper mapper =
7             getSqlSession().getMapper(StudentMapper.class);
8         mapper.insertAddress(student.getAddress());
9         //Custom logic
10        mapper.insertStudent(student);
11    }
12 }

```

XML Code

```

1 <bean id="studentMapper" class="com.mybatis3.dao.StudentMapperImpl">
2     <property name="sqlSessionFactory" ref="sqlSessionFactory" />
3 </bean>
4

```

在以上的这些方式中，我们注入了 `SqlSession` 对象，获取 `Mapper` 实例，然后执行映射语句。这里 Spring 会为我们提供一个线程安全的 `SqlSession` 对象，以及当方法结束后关闭 `SqlSession` 对象。

然而，MyBatis-Spring 模块提供了更好的方式，我们可以不通过 `SqlSession` 获取映射器 `Mapper`，直接注射 `Sql`

映射器 Mapper bean。我们下节将讨论它。

5.3 使用映射器

我们可以使用 MapperFactoryBean 将映射器 Mapper 接口配置成 Spring bean 实体。如下所示：

Java Code

```
1 public interface StudentMapper
2 {
3     @Select("select stud_id as studId, name, email, phone from
4             students where stud_id=#{id}")
5     Student findStudentById(Integer id);
6 }
```

XML Code

```
1 <bean id="studentMapper" class="org.mybatis.spring.mapper. MapperFactoryBean">
2     <property name="mapperInterface" value="com.mybatis3.mappers. StudentMapper" />
3     <property name="sqlSessionFactory" ref="sqlSessionFactory" />
4 </bean>
5
```

现在 StudentMapper bean 实体对象可以被注入到任意的 Spring bean 实体对象中，并调用映射语句方法，如下所示：

Java Code

```
1 public class StudentService
2 {
3     private StudentMapper studentMapper;
4     public void setStudentMapper (StudentMapper studentMapper)
5     {
6         this.studentMapper = studentMapper;
7     }
8     public void createStudent(Student student)
9     {
10        this.studentMapper.insertStudent(student);
11    }
12 }
```

XML Code

```
1 <bean id="studentService" class="com.mybatis3.services. StudentService">
2     <property name="studentMapper" ref="studentMapper" />
3 </bean>
```

分别配置每一个映射器 Mapper 接口是一个非常单调的过程。我们可以使用 `MapperScannerConfigurer` 来扫描包 (package) 中的映射器 Mapper 接口，并自动地注册。

XML Code

```
1 <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
2   <property name="basePackage" value="com.mybatis3.mappers" />
3 </bean>
4
```

如果映射器 Mapper 接口在不同的包(package)中，你可以为 `basePackage` 属性指定一个以逗号分隔的包名列表。

MyBatis-Spring-1.2.0 介绍了两种新的扫描映射器 Mapper 接口的方法：

- 使用 `<mybatis:scan>` 元素
- 使用 `@MapperScan` 注解 (需 Spring 3.1+ 版本)

5.3.1 <mybatis:scan />

`<mybatis:scan>` 元素将在特定的以逗号分隔的包名列表中搜索映射器 Mapper 接口。使用这个新的 MyBatis-Spring 命名空间你需要添加以下的 schema 声明：

XML Code

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
4   xsi:schemaLocation="http://www.springframework.org/schema/beans
5     http://www.springframework.org/schema/beans/spring-beans.xsd
6     http://mybatis.org/schema/mybatis-spring
7     http://mybatis.org/schema/mybatis-spring.xsd">
8   <mybatis:scan base-package="com.mybatis3.mappers" />
9 </beans>
```

`<mybatis:scan>` 元素提供了下列的属性来自定义扫描过程：

- **annotation**：扫描器将注册所有的在 `base-package` 包内并且匹配指定注解的映射器 Mapper 接口。
- **factory-ref**：当 Spring 上下文中有多个 `SqlSessionFactory` 实例时，需要指定某一特定的 `SqlSessionFactory` 来创建映射器 Mapper 接口。正常情况下，只有应用程序中有一个以上的数据源才会使用。
- **marker-interface**：扫描器将注册在 `base-package` 包中的并且继承了特定的接口类的映射器 Mapper 接口
- **template-ref**：当 Spring 上下文中有多个 `SqlSessionTemplate` 实例时，需要指定某一特定的 `SqlSessionTemplate` 来创建映射器 Mapper 接口。正常情况下，只有应用程序中有一个以上的数据源才会使用。

- **name-generator**: BeannameGenerator 类的完全限定类名，用来命名检测到的组件。

5.3.2 MapperScan

Spring 框架 3.x+版本支持使用@Configuration 和@Bean 注解来提供基于 Java 的配置。如果你倾向于使用基于 Java 的配置，你可以使用@MapperScan 注解来扫描映射器 Mapper 接口。@MapperScan 和<mybatis:scan/>工作方式相同，并且也提供了对应的自定义选项。

Java Code

```

1 @Configuration
2 @MapperScan("com.mybatis3.mappers")
3 public class AppConfig
4 {
5     @Bean
6     public DataSource dataSource()
7     {
8         return new PooledDataSource("com.mysql.jdbc.Driver",
9                                     "jdbc:mysql://localhost:3306/elearning", "root", "admin");
10    }
11    @Bean
12    public SqlSessionFactory sqlSessionFactory() throws Exception
13    {
14        SqlSessionFactoryBeansessionFactory = new
15        SqlSessionFactoryBean();
16        sessionFactory.setDataSource(dataSource());
17        return sessionFactory.getObject();
18    }
19 }

```

@MapperScan 注解有以下属性供自定义扫描过程使用：

- **annotationClass**: 扫描器将注册所有的在 base-package 包内并且匹配指定注解的映射器 Mapper 接口。
- **markerInterface**: 扫描器将注册在 base-package 包中的并且继承了特定的接口类的映射器 Mapper 接口
- **sqlSessionFactoryRef**: 当 Spring 上下文中有有一个以上的 SqlSesssionFactory 时，用来指定特定 SqlSessionFactory
- **sqlSessionTemplateRef**: 当 Spring 上下文中有有一个以上的 sqlSessionTemplate 时，用来指定特定 sqlSessionTemplate
- **nameGenerator**: BeanNameGenerator 类用来命名在 Spring 容器内检测到的组件。
- **basePackageClasses**: basePackages() 的类型安全的替代品。包内的每一个类都会被扫描。
- **basePackages**: 扫描器扫描的基包，扫描器会扫描内部的 Mapper 接口。注意包内的至少有一个方法声明的才会被注册。具体类将会被忽略。



与注入 Sqlsession 相比，更推荐使用注入 Mapper，因为它摆脱了对 MyBatis API 的依赖。

5.4 使用 Spring 进行事务管理

只使用 MyBatis，你需要写事务控制相关代码，如提交或者回退数据库操作。

Java Code

```
1 public Student createStudent(Student student)
2 {
3     SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().
4         openSession();
5     try
6     {
7         StudentMapper mapper =
8             sqlSession.getMapper(StudentMapper.class);
9         mapper.insertAddress(student.getAddress());
10        mapper.insertStudent(student);
11        sqlSession.commit();
12        return student;
13    }
14    catch (Exception e)
15    {
16        sqlSession.rollback();
17        throw new RuntimeException(e);
18    }
19    finally
20    {
21        sqlSession.close();
22    }
23 }
```

我们可以使用 Spring 的基于注解的事务处理机制来避免书写上述的每个方法中控制事务的冗余代码。

为了能使用 Spring 的事务管理功能，我们需要在 Spring 应用上下文中配置 TransactionManager bean 实体对象：

XML Code

```
1 <bean id="transactionManager"
2     class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
3     <property name="dataSource" ref="dataSource" />
4 </bean>
5
```

事务管理器引用的 dataSource 和 SqlSessionFactory bean 使用的 dataSource 相同。

在 Spring 中使用基于注解的事务管理特性，如下：

XML Code

```
1 <tx:annotation-driven transaction-manager="transactionManager"/>
```

现在你可以在 Spring service bean 上使用 `@Transactional` 注解，表示在此 service 中的每一个方法都应该在一个事务中运行。如果方法成功运行完毕，Spring 会提交操作。如果有运行期异常发生，则会执行回滚操作。另外，Spring 会将 MyBatis 的异常转换成合适的 `DataAccessExceptions`，这样会为特定错误上提供额外的信息。

Java Code

```
1 @Service
2 @Transactional
3 public class StudentService
4 {
5     @Autowired
6     private StudentMapper studentMapper;
7     public Student createStudent(Student student)
8     {
9         studentMapper.insertAddress(student.getAddress());
10        if(student.getName().equalsIgnoreCase(""))
11        {
12            throw new RuntimeException("Student name should not be
13                                     empty.");
14        }
15        studentMapper.insertStudent(student);
16        return student;
17    }
18 }
```

下面是一个 Spring 的 applicationContext.xml 完成配置：

XML Code

```
1 <beans>
2     <context:annotation-config />
3     <context:component-scan base-package="com.mybatis3" />
4     <context:property-placeholder location="classpath:application.properties" />
5     <tx:annotation-driven transaction-manager="transactionManager" />
6     <bean id="transactionManager"
7         class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
8         <property name="dataSource" ref="dataSource" />
9     </bean>
10    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
11        <property name="basePackage" value="com.mybatis3.mappers" />
12    </bean>
13    <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
14        <constructor-arg index="0" ref="sqlSessionFactory" />
15    </bean>
```

```

15 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
16     <property name="dataSource" ref="dataSource" />
17     <property name="typeAliases"
18         value="com.mybatis3.domain.Student,com.mybatis3.domain.Tutor" />
19     <property name="typeAliasesPackage" value="com.mybatis3.domain" />
20     <property name="typeHandlers" value="com.mybatis3.typehandlers.PhoneTypeHandler" />
21     <property name="typeHandlersPackage" value="com.mybatis3.typehandlers" />
22     <property name="mapperLocations" value="classpath*:com/mybatis3/**/*.xml" />
23 </bean>
24 <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
25     <property name="driverClassName" value="${jdbc.driverClassName}"></property>
26     <property name="url" value="${jdbc.url}"></property>
27     <property name="username" value="${jdbc.username}"></property>
28     <property name="password" value="${jdbc.password}"></property>
    </bean>
</beans>

```

现在让我们写一个独立的测试客户端来测试 StudentService , 如下 :

Java Code

```

1 @RunWith(SpringJUnit4ClassRunner.class)
2 @ContextConfiguration(locations = "classpath:applicationContext.xml"
3     )
4 public class StudentServiceTest
5 {
6     @Autowired
7     private StudentService studentService;
8     @Test
9     public void testCreateStudent()
10    {
11        Address address = new Address(0, "Quaker Ridge
12                                Rd.", "Bethel", "Brooklyn", "06801", "USA");
13        Student stud = new Student();
14        long ts = System.currentTimeMillis();
15        stud.setName("stud_" + ts);
16        stud.setEmail("stud_" + ts + "@gmail.com");
17        stud.setAddress(address);
18        Student student = studentService.createStudent(stud);
19        assertNotNull(student);
20        assertEquals("stud_" + ts, student.getName());
21        assertEquals("stud_" + ts + "@gmail.com", student.getEmail());
22        System.err.println("CreatedStudent: " + student);
23    }
24    @Test(expected = DataAccessException.class)
25    public void testCreateStudentForException()

```

```

26     {
27         Address address = new Address(0, "Quaker Ridge
28                                     Rd.", "Bethel", "Brooklyn", "06801", "USA");
29         Student stud = new Student();
30         long ts = System.currentTimeMillis();
31         stud.setName("Timothy");
32         stud.setEmail("stud_" + ts + "@gmail.com");
33         stud.setAddress(address);
34         studentService.createStudent(stud);
35         fail("You should not reach here");
36     }
37 }

```

这里在 `testCreateStudent()` 方法中,我们为 `Address` 和 `Student` 赋上了合适的数据,所以 `Address` 和 `Student` 会被分别插入到表 `ADDRESSES` 和 `STUDENTS` 中。在 `testCreateStudentForException()` 方法我们设置了名字为 `Timothy`,该名称在数据库中已经存在了,所以当你尝试将此 `student` 记录插入到数据库中,MySQL 会抛出一个 `UNIQUE KEY` 冲突的异常,Spring 会将此异常转换成 `DataAccessException` 异常,并且将插入 `ADDRESSES` 表中的数据回滚 (rollback) 掉。

5.5 总结

在本章中我们学习了怎样将 MyBatis 与 Spring 框架集成。我们还学习了怎样安装 Spring 类库并且在 Spring 的应用上下文 `ApplicationContext` 上注册 MyBatis bean 实体对象。我们还看到怎样配置和注入 `SqlSession` 和 `Mapper` bean 实体对象以及调用映射语句。我们还学习了利用 Spring 基于注解的事务处理机制来使用 MyBatis。

你已经读完本书,祝贺你!现在,你应该知道怎样高效地使用 MyBatis 与数据库工作。你学会了怎样发挥你的 Java 和 SQL 技巧的优势使 MyBatis 更富有成效。你知道了怎样以更清晰的方式使用 MyBatis 写出数据持久化代码,不用管被 MyBatis 框架处理的所有底层细节。另外,你学会了怎样在最流行的依赖注入框架-Spring 中使用 MyBatis。

MyBatis 框架非常易于使用,但它提供了强大的特性,因此它对于基于 Java 的项目而言,是一个非常好的数据库持久化解决方案。MyBatis 也提供了一些工具如 MyBatis Generator(<http://www.mybatis.org/generator/>),可以被用来从已经存在的数据库 schema 中,产生持久化代码如数据库实体 (database entities),映射器 `Mapper` 接口, `MapperXML` 配置文件,使 MyBatis 入门非常方便。另外,MyBatis 还有它的姊妹项目如 MyBatis.NET 和 MyBatis-Scala,分别为 .NET 和 Scala 编程语言提供了一样强大的特性。

MyBatis 随着每一个版本的发布,增加了一些特性,正变得越来越好。想了解更多的新特性,你可以访问 MyBatis 官方网站 <https://code.google.com/p/mybatis/>。订阅 MyBatis 的使用者邮件列表是一个不错的想法。我们祝你一切顺利,编码快乐!(We wish you all the best, and happy coding!)