入门

安装

要使用 MyBatis，您只需在类路径中包含 mybatis-x.x.x.jar 🎈 文件。

如果您使用 Maven，只需将以下依赖项添加到您的 pom.xml

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>x.x.x</version>
</dependency>
```

从 XML 构建 SqlSessionFactory

每个 MyBatis 应用程序都围绕一个 SqlSessionFactory 实例进行。可以通过使用 SqlSessionFactoryBuilder 获取 SqlSessionFactory 实例。SqlSessionFactoryBuilder 可以从 XML 配置文件或从 Configuration 类的自定义准备实例构建 SqlSessionFactory 实例。

从 XML 文件构建 SqlSessionFactory 实例非常简单。建议您为此配置使用类路径资源，但您可以使用任何 InputStream 实例，包括从实际文件路径或 `file://` URL 创建的实例。MyBatis 包含一个名为 Resources 的实用程序类，其中包含许多方法，可简化从类路径和其他位置加载资源。

```
String resource = "org/mybatis/example/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory =
  new SqlSessionFactoryBuilder().build(inputStream);
```

配置 XML 文件包含 MyBatis 系统核心的设置，包括用于获取数据库连接实例的数据源，以及用于确定事务范围和控制方式的事务管理器。XML 配置文件的完整详细信息可以在本文档后面找到，但这里有一个简单的示例

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.ac.cn//DTD Config 3.0//EN"
  "https://mybatis.ac.cn/dtd/mybatis-3-config.dtd">
<configuration>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

虽然 XML 配置文件还有很多内容，但上面的示例指出了最关键的部分。请注意 XML 头，需要验证 XML 文档。environment 元素的主体包含事务管理和连接池的环境配置。mappers 元素包含映射器列表 - 包含 SQL 代码和映射定义的 XML 文件和/或带注释的 Java 接口类。

不使用 XML 构建 SqlSessionFactory

如果您更愿意直接从 Java 而不是 XML 构建配置，或创建自己的配置生成器，MyBatis 提供了一个完整的 Configuration 类，该类提供与 XML 文件相同的所有配置选项。

```
DataSource dataSource = BlogDataSourceFactory.getBlogDataSource();
TransactionFactory transactionFactory =
  new JdbcTransactionFactory();
Environment environment =
  new Environment("development", transactionFactory, dataSource);
Configuration configuration = new Configuration(environment);
configuration.addMapper(BlogMapper.class);
SqlSessionFactory sqlSessionFactory =
  new SqlSessionFactoryBuilder().build(configuration);
```

请注意，在这种情况下，配置正在添加一个映射器类。映射器类是包含 SQL 映射注释的 Java 类，可避免使用 XML 映射。但是，由于 Java 注释存在一些限制，并且某些 MyBatis 映射很复杂，因此对于最高级的映射（例如嵌套连接映射），仍然需要 XML 映射。出于这个原因，如果 MyBatis 存在（在这种情况下，将根据 BlogMapper.class 的类路径和名称加载 BlogMapper.xml），MyBatis 将自动查找并加载对等 XML 文件。稍后对此进行更多介绍。

从 SqlSessionFactory 获取 SqlSession

现在您有了 SqlSessionFactory，顾名思义，您可以获取 SqlSession 的实例。SqlSession 包含执行针对数据库的 SQL 命令所需的所有方法。您可以直接针对 SqlSession 实例执行映射的 SQL 语句。例如

```
try (SqlSession session = sqlSessionFactory.openSession()) {
  Blog blog = session.selectOne(
    "org.mybatis.example.BlogMapper.selectBlog", 101);
}
```

虽然此方法有效，并且对于以前版本的 MyBatis 用户来说很熟悉，但现在有了一种更简洁的方法。使用正确描述给定语句的参数和返回值的接口（例如 BlogMapper.class），您现在可以执行更简洁、类型更安全的代码，而无需容易出错的字符串文字和强制转换。

例如

```
try (SqlSession session = sqlSessionFactory.openSession()) {
  BlogMapper mapper = session.getMapper(BlogMapper.class);
  Blog blog = mapper.selectBlog(101);
}
```

现在让我们探讨这里到底执行了什么。

探索映射的 SQL 语句

此时，您可能想知道 SqlSession 或映射器类到底执行了什么。映射的 SQL 语句是一个很大的主题，并且该主题可能会占据本文档的大部分内容。但为了让您了解到底运行了什么，这里有一些示例。

在以上任何示例中，语句都可以由 XML 或注释定义。我们先来看看 XML。MyBatis 提供的全部功能集可以通过使用基于 XML 的映射语言来实现，该语言多年来使 MyBatis 广受欢迎。如果您以前使用过 MyBatis，那么您会熟悉这个概念，但 XML 映射文档已经有了许多改进，稍后会变得清晰。以下是一个基于 XML 的映射语句的示例，它将满足上述 SqlSession 调用。

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

虽然对于这个简单的示例来说，这看起来有很多开销，但实际上它非常轻量。您可以在一个映射器 XML 文件中定义任意数量的映射语句，因此您可以充分利用 XML 头和 doctype 声明。文件的其余部分非常容易理解。它为映射语句"selectBlog"定义了一个名称，在命名空间"org.mybatis.example.BlogMapper"中，这将允许您通过指定"org.mybatis.example.BlogMapper.selectBlog"的完全限定名称来调用它，正如我们在以下示例中所做的那样

```
Blog blog = session.selectOne(
  "org.mybatis.example.BlogMapper.selectBlog", 101);
```

注意这与调用完全限定 Java 类的某个方法非常相似，并且这里有一个原因。此名称可直接映射到与命名空间同名的 Mapper 类，该类具有与映射的 select 语句相匹配的名称、参数和返回类型的方法。这允许您非常简单地针对 Mapper 接口调用该方法，如您在上面看到的，但这里在以下示例中再次展示了该方法

```
BlogMapper mapper = session.getMapper(BlogMapper.class);
Blog blog = mapper.selectBlog(101);
```

第二种方法有很多优点。首先，它不依赖字符串文本，因此更安全。其次，如果您的 IDE 具有代码完成功能，则在浏览映射的 SQL 语句时可以利用该功能。

有关命名空间的说明。

命名空间在以前版本的 MyBatis 中是可选的，这令人困惑且无益。现在命名空间是必需的，并且除了使用更长的完全限定名称来隔离语句之外，还有其他用途。

命名空间启用您在此处看到的接口绑定，即使您认为今天不会使用它们，您也应该遵循此处列出的这些做法，以防您改变主意。使用命名空间一次，并将其放入适当的 Java 包命名空间中，将清理您的代码并从长远来看提高 MyBatis 的可用性。

名称解析：为了减少输入量，MyBatis 对所有命名的配置元素（包括语句、结果映射、缓存等）使用以下名称解析规则。

- 完全限定名称（例如"com.mypackage.MyMapper.selectAllThings"）将被直接查找并使用（如果找到）。
- 短名称（例如"selectAllThings"）可用于引用任何明确的条目。但是，如果存在两个或更多（例如"com.foo.selectAllThings 和 com.bar.selectAllThings"），那么您将收到一个错误报告，指出短名称不明确，因此必须完全限定。

对于像 BlogMapper 这样的 Mapper 类，还有一个技巧。它们的映射语句根本不需要使用 XML 映射。相反，它们可以使用 Java 注解。例如，可以消除上面的 XML 并用以下内容替换

```
package org.mybatis.example;
public interface BlogMapper {
  @Select("SELECT * FROM blog WHERE id = #{id}")
  Blog selectBlog(int id);
}
```

对于简单的语句，注解要干净得多，但是，对于更复杂的语句，Java 注解既有限制又混乱。因此，如果您必须执行任何复杂操作，那么最好使用 XML 映射语句。

由您和您的项目团队决定哪种方法适合您，以及您的映射语句以一致的方式定义对您有多重要。也就是说，您永远不会被锁定到单一方法。您可以非常轻松地将基于注解的映射语句迁移到 XML，反之亦然。

范围和生命周期

理解我们到目前为止讨论过的各种作用域和生命周期类非常重要。使用不当会导致严重的并发问题。

对象生命周期和依赖注入框架

依赖注入框架可以创建线程安全、事务性的 SqlSession 和映射器，并将它们直接注入到您的 Bean 中，以便您可以忘记它们的生命周期。您可能希望了解 MyBatis-Spring 或 MyBatis-Guice 子项目，以了解有关将 MyBatis 与 DI 框架结合使用的更多信息。

SqlSessionFactoryBuilder

此类可以实例化、使用和丢弃。创建 SqlSessionFactory 后无需保留它。因此，SqlSessionFactoryBuilder 实例的最佳作用域是方法作用域（即局部方法变量）。您可以重复使用 SqlSessionFactoryBuilder 来构建多个 SqlSessionFactory 实例，但最好不要保留它，以确保所有 XML 解析资源都释放出来以用于更重要的事情。

SqlSessionFactory

创建后，SqlSessionFactory 应在您的应用程序执行期间存在。几乎没有理由处置它或重新创建它。最佳做法是在应用程序运行中不要多次重建 SqlSessionFactory。这样做应被视为"坏味道"。因此，SqlSessionFactory 的最佳作用域是应用程序作用域。这可以通过多种方式实现。最简单的方法是使用单例模式或静态单例模式。

SqlSession

每个线程都应有自己的 SqlSession 实例。SqlSession 实例不应共享，并且不是线程安全的。因此，最佳作用域是请求或方法作用域。切勿在静态字段或类的实例字段中保留对 SqlSession 实例的引用。切勿在任何类型的受管作用域（例如 Servlet 框架的 HttpSession）中保留对 SqlSession 的引用。如果您使用任何类型的 Web 框架，请考虑让 SqlSession 遵循与 HTTP 请求类似的作用域。换句话说，在收到 HTTP 请求后，您可以打开一个 SqlSession，然后在返回响应后，您可以关闭它。关闭会话非常重要。您应始终确保它在 finally 块中关闭。以下是确保 SqlSession 关闭的标准模式

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

在整个代码中始终如一地使用此模式将确保所有数据库资源都正确关闭。

映射器实例

映射器是您创建的用于绑定到映射语句的接口。映射器接口的实例从 SqlSession 获取。因此，从技术上讲，任何映射器实例的最广泛范围与从其请求的 SqlSession 相同。但是，映射器实例的最佳范围是方法范围。也就是说，它们应该在使用它们的方法中请求，然后被丢弃。它们不需要显式关闭。虽然在整个请求中保留它们并不是问题，类似于 SqlSession，但您可能会发现在此级别管理太多资源会很快失控。保持简单，将映射器保留在方法范围内。以下示例演示了此实践。

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