1. 定义：一款持久层的框架，简化jdbc的开发
2. 持久层：负责将数据保存到数据库的那一层
3. JavaEE三层架构：表现层、业务层、持久层
4. 使用JDBC带来的问题：硬编码（注册编码，sql语句），操作繁琐（手动设置参数，手动封装结果集）
5. 快速入门：

查询一个user表

(1)导入坐标， <dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis</artifactId>  
 <version>3.5.5</version>  
 </dependency>  
  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.33</version>  
 </dependency>  
  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>3.8.1</version>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-core</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.36</version>  
 </dependency>  
</dependencies>

(2)创建SqlSessionFactory的xml表

文本

AI 生成的内容可能不正确。

(3)创建日志文件，logback.xml

<?xml version="1.0" encoding="UTF-8"?>  
<configuration debug="false">  
  
 <!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径-->  
 <property name="LOG\_HOME" value="/home" />  
  
 <!--控制台日志， 控制台输出 -->  
 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度,%msg：日志消息，%n是换行符-->  
 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>  
 </encoder>  
 </appender>  
  
 <!--文件日志， 按照每天生成日志文件 -->  
 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <!--日志文件输出的文件名-->  
 <FileNamePattern>${LOG\_HOME}/TestWeb.log.%d{yyyy-MM-dd}.log</FileNamePattern>  
 <!--日志文件保留天数-->  
 <MaxHistory>30</MaxHistory>  
 </rollingPolicy>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->  
 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>  
 </encoder>  
 <!--日志文件最大的大小-->  
 <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">  
 <MaxFileSize>10MB</MaxFileSize>  
 </triggeringPolicy>  
 </appender>  
  
 <!-- show parameters for hibernate sql 专为 Hibernate 定制 -->  
 <logger name="org.hibernate.type.descriptor.sql.BasicBinder" level="TRACE" />  
 <logger name="org.hibernate.type.descriptor.sql.BasicExtractor" level="DEBUG" />  
 <logger name="org.hibernate.SQL" level="DEBUG" />  
 <logger name="org.hibernate.engine.QueryParameters" level="DEBUG" />  
 <logger name="org.hibernate.engine.query.HQLQueryPlan" level="DEBUG" />  
  
 <!--myibatis log configure-->  
 <logger name="com.apache.ibatis" level="TRACE"/>  
 <logger name="java.sql.Connection" level="DEBUG"/>  
 <logger name="java.sql.Statement" level="DEBUG"/>  
 <logger name="java.sql.PreparedStatement" level="DEBUG"/>  
  
 <!-- 日志输出级别 -->  
 <root level="DEBUG">  
 <appender-ref ref="STDOUT" />  
 <appender-ref ref="FILE"/>  
 </root>  
</configuration>

（4）创建映射文件，即写sql的表

文本

AI 生成的内容可能不正确。

（5）创建接收数据的类

文本

AI 生成的内容可能不正确。

（6）编写对应代码

文本

AI 生成的内容可能不正确。

1. Mapper代理开发
2. 定义与sql映射文件同名的Mapper接口，并将Mapper接口和SQL映射文件放在同一目录下，具体的实现方法：在resources文件中创建一个和java中一样名称的文件，并将对应的sql的映射文件放入，然后java中的那个同名文件中创建一个接口，并将resources中同名文件中sql映射文件中的sql语句的id作为方法名放在接口中，这样映射文件就会和接口放在同一文件夹中，并且将sql映射文件的namespace改为该其所在文件，同时conflg中的Mappers可以改为package的形式，这样就可以一次性导入一个接口中的所有sql映射文件

图形用户界面, 文本

AI 生成的内容可能不正确。

电脑萤幕截图

AI 生成的内容可能不正确。

截图里有图片

AI 生成的内容可能不正确。

电脑的屏幕截图

AI 生成的内容可能不正确。

电脑屏幕截图

AI 生成的内容可能不正确。

1. 核心配置文件
2. environment：配置库，可以同时连接多个数据库。
3. typeAliases：可以给包含接收类的对象其别名，简化导入文件路径的名称
4. 查询和结果映射
5. 数据库表的字段名和实体类的属性名不一致的解决办法：第一种方法就是给在sql语段中给字段名起别名，使之与实体类的属性名对应；第二种方法是使用使用resultMap映射进行解决，即一种动态的起别名，这样更加灵活，不需要每次都重新命名，不过sql语句映射中的resultType要改为resultMap
6. 详细查询：由于要接受变量，所以要使用模糊查询，在总体上和查询全部基本一致，不过改变的是sql映射要发生改变，接收变量也要改变

文本

AI 生成的内容可能不正确。

屏幕的截图

AI 生成的内容可能不正确。

1. 静态条件查询

散装参数

文本

AI 生成的内容可能不正确。

电脑萤幕的截图

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

对象参数

文本

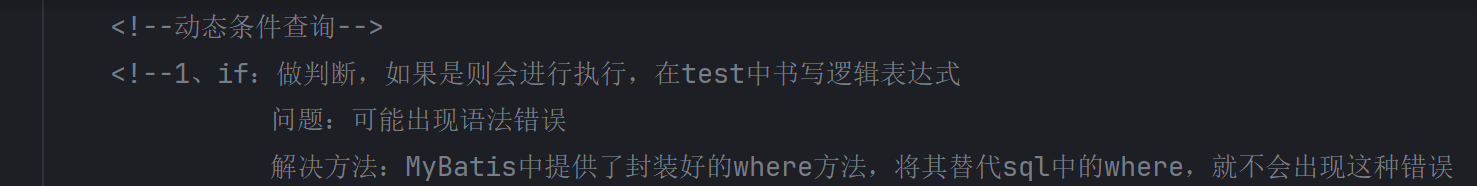
AI 生成的内容可能不正确。

Map集合查询

文本

AI 生成的内容可能不正确。

1. 动态条件查询



屏幕的截图

AI 生成的内容可能不正确。

图形用户界面

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

1. 添加数据

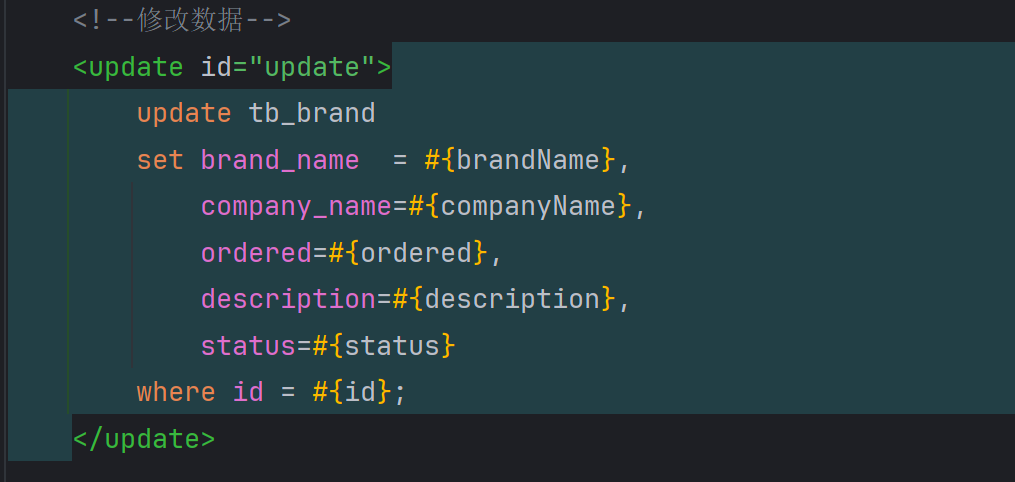
文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

1. 修改数据
2. 全部进行修改



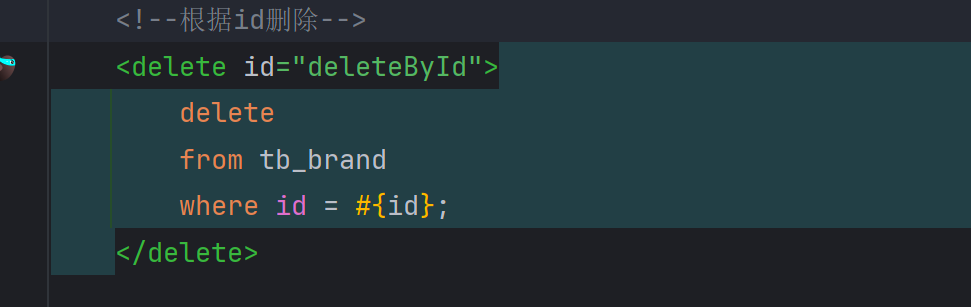
1. 动态修改

文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

1. 删除
2. 根据id删除：

文本

AI 生成的内容可能不正确。

1. 批量删除

屏幕的截图

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

1. 参数传递
2. 多个参数：封装为Map集合，可以使用Param进行注解，替换集合中默认的键名arg

电脑屏幕截图

AI 生成的内容可能不正确。

1. 单个参数：

POJO类：直接使用，属性名和参数占位符一致

Map集合：直接使用，键名和参数占位符一致

Collection：封装为Map集合，使用@Param注解,

Map.put(“arg0”,collection集合);

Map.put(“collection”,collection集合);

List集合：封装为Map集合，使用@Param注解

Map.put(“arg0”,参数值1);

Map.put(“collection”,List集合);

Map.put(“list”,List集合);

Array集合：封装为Map集合，使用@param注解

Map.put(“arg1”,数组);

Map.put(“array”,数组);

其它类：直接进行使用

1. 注解完成增删改查
2. 适用场景：实现相对简单的sql语句
3. 自动关联：采用注解的方式来书写 SQL 语句时，确实没有像 XML 配置文件中的 id 一样明确指定 SQL 语句的标识符。不过，MyBatis 会根据方法名来关联对应的 SQL 注解。也就是说，SQL 注解与方法是绑定的，MyBatis 通过方法的调用来决定执行哪个 SQL 注解。
4. 一一对应：一个方法前面不能够同时出现多个相同或不用类型的注解，一般一个方法前只会书写一个注解，提供给方法进行调用。
5. 接收不到参数：由于注解本身并不能够直接接收到resultMap，因此，会出现参数名和数据库中属性名不一致后无法接收的问题，通常可以使用@Results来进行解决