

# Day02

---

## MyBatis 3.5.x

### MyBatis API

#### 1. Resources

##### 1.1 主要信息描述

##### 1.2 静态方法描述

#### 2. SqlSessionFactoryBuilder

##### 2.1 主要信息描述

##### 2.2 构造方法描述

##### 2.3 方法描述

#### 3. SqlSessionFactory

##### 3.1 主要信息描述

##### 3.2 方法描述

#### 4. SqlSession

##### 4.1 主要信息描述

##### 4.2 方法描述

### MyBatis 核心配置文件

#### 事务

默认自动提交为 false

开启自动提交

### MyBatis 的 CRUD

#### 插入记录

##### 1. 修改 dao 层接口

##### 2. 实现 dao 层接口的方法

##### 3. 对新增的方法做单元测试

#### 更新记录

##### 1. 修改 dao 层接口

##### 2. 实现 dao 层接口的方法

##### 3. 对方法做单元测试

删除记录

1. 修改 dao 层接口
2. 实现 dao 层接口的方法
3. 对方法做单元测试

课堂作业

MyBatis 工具类

课堂作业

传参

一个参数

多个参数

课堂作业

课后作业

## MyBatis 3.5.x

## MyBatis API

通过快速入门，我们应该知道 MyBatis 使用的大概流程了，接下来我们将开始熟悉 MyBatis 中的 API。

### 1. Resources

Resources 是 MyBatis 提供给我们用于加载资源的工具类，通常用于加载核心配置文件。

#### 1.1 主要信息描述

程序包 `org.apache.ibatis.io`

#### 类 **Resources**

`java.lang.Object`  
`org.apache.ibatis.io.Resources`

---

```
public class Resources
extends java.lang.Object
```

A class to simplify access to resources through the classloader.

## 1.2 静态方法描述

常用方法是静态方法：`getResourceAsStream()`

修饰符和类型	方法	说明
static java.lang.Class<?>	<code>classForName(java.lang.String className)</code>	Loads a class
static java.nio.charset.Charset	<code>getCharset()</code>	
static java.lang.ClassLoader	<code>getDefaultClassLoader()</code>	Returns the default classloader (may be null).
static java.io.File	<code>getResourceAsFile(java.lang.ClassLoader loader, java.lang.String resource)</code>	Returns a resource on the classpath as a File object
static java.io.File	<code>getResourceAsFile(java.lang.String resource)</code>	Returns a resource on the classpath as a File object
static java.util.Properties	<code>getResourceAsProperties(java.lang.ClassLoader loader, java.lang.String resource)</code>	Returns a resource on the classpath as a Properties object
static java.util.Properties	<code>getResourceAsProperties(java.lang.String resource)</code>	Returns a resource on the classpath as a Properties object
static java.io.Reader	<code>getResourceAsReader(java.lang.ClassLoader loader, java.lang.String resource)</code>	Returns a resource on the classpath as a Reader object
static java.io.Reader	<code>getResourceAsReader(java.lang.String resource)</code>	Returns a resource on the classpath as a Reader object
static java.io.InputStream	<code>getResourceAsStream(java.lang.ClassLoader loader, java.lang.String resource)</code>	Returns a resource on the classpath as a Stream object
static java.io.InputStream	<code>getResourceAsStream(java.lang.String resource)</code>	Returns a resource on the classpath as a Stream object
static java.net.URL	<code>getResourceURL(java.lang.ClassLoader loader, java.lang.String resource)</code>	Returns the URL of the resource on the classpath
static java.net.URL	<code>getResourceURL(java.lang.String resource)</code>	Returns the URL of the resource on the classpath
static java.util.Properties	<code>getURLAsProperties(java.lang.String urlString)</code>	Gets a URL as a Properties object
static java.io.Reader	<code>getURLAsReader(java.lang.String urlString)</code>	Gets a URL as a Reader
static java.io.InputStream	<code>getURLAsStream(java.lang.String urlString)</code>	Gets a URL as an input stream
static void	<code>setCharset(java.nio.charset.Charset charset)</code>	
static void	<code>setDefaultClassLoader(java.lang.ClassLoader defaultClassLoader)</code>	Sets the default classloader

## 2. SqlSessionFactoryBuilder

`SqlSessionFactoryBuilder` 的作用是创建 `SqlSessionFactory` 对象（也就是创建 `SqlSession` 对象的工厂对象）。

### 2.1 主要信息描述

程序包 `org.apache.ibatis.session`

类 **`SqlSessionFactoryBuilder`**

java.lang.Object  
org.apache.ibatis.session.SqlSessionFactoryBuilder

public class **SqlSessionFactoryBuilder**  
extends java.lang.Object

Builds `SqlSession` instances.

### 2.2 构造方法描述

## 构造器概要

### 构造器

#### 构造器

**SqlSessionFactoryBuilder()**

## 2.3 方法描述

### 方法概要

所有方法	实例方法	具体方法
修饰符和类型	方法	
SqlSessionFactory	build	build(java.io.InputStream inputStream)
SqlSessionFactory	build	build(java.io.InputStream inputStream, java.lang.String environment)
SqlSessionFactory	build	build(java.io.InputStream inputStream, java.lang.String environment, java.util.Properties properties)
SqlSessionFactory	build	build(java.io.InputStream inputStream, java.util.Properties properties)
SqlSessionFactory	build	build(java.io.Reader reader)
SqlSessionFactory	build	build(java.io.Reader reader, java.lang.String environment)
SqlSessionFactory	build	build(java.io.Reader reader, java.lang.String environment, java.util.Properties properties)
SqlSessionFactory	build	build(java.io.Reader reader, java.util.Properties properties)
SqlSessionFactory	build	build(Configuration config)

## 3. SqlSessionFactory

### 3.1 主要信息描述

SqlSessionFactory 通过一个连接或者数据源的信息来创建 SqlSession 对象。

程序包 org.apache.ibatis.session

### 接口 SqlSessionFactory

所有已知实现类:

DefaultSqlSessionFactory, SqlSessionManager

---

public interface **SqlSessionFactory**

Creates an **SqlSession** out of a connection or a DataSource

### 3.2 方法描述

## 方法概要

所有方法

实例方法

抽象方法

修饰符和类型

方法

<b>Configuration</b>	<b>getConfiguration()</b>
<b>SqlSession</b>	<b>openSession()</b>
<b>SqlSession</b>	<b>openSession(boolean autoCommit)</b>
<b>SqlSession</b>	<b>openSession(java.sql.Connection connection)</b>
<b>SqlSession</b>	<b>openSession(ExecutorType execType)</b>
<b>SqlSession</b>	<b>openSession(ExecutorType execType, boolean autoCommit)</b>
<b>SqlSession</b>	<b>openSession(ExecutorType execType, java.sql.Connection connection)</b>
<b>SqlSession</b>	<b>openSession(ExecutorType execType, TransactionIsolationLevel level)</b>
<b>SqlSession</b>	<b>openSession(TransactionIsolationLevel level)</b>

## 4. SqlSession

### 4.1 主要信息描述

SqlSession 对象用于执行 SQL、接口代理和管理事务。

我们可以看到官方API对这个类的描述是日常使用 MyBatis 中的主要接口，可见其强大。

是我们重点需要了解和学习内容。

程序包 org.apache.ibatis.session

#### 接口 SqlSession

所有超级接口:

java.lang.AutoCloseable, java.io.Closeable

所有已知实现类:

DefaultSqlSession, SqlSessionManager

```
public interface SqlSession
extends java.io.Closeable
```

The primary Java interface for working with MyBatis. Through this interface you can execute commands, get mappers and manage transactions.

### 4.2 方法描述

常用方法:

close、commit、delete、getMapper、insert、rollback、selectList、selectOne、update

## 方法概要

所有方法	实例方法	抽象方法
修饰符和类型	方法	说明
void	<code>clearCache()</code>	Clears local session cache.
void	<code>close()</code>	Closes the session.
void	<code>commit()</code>	Flushes batch statements and commits database connection.
void	<code>commit(boolean force)</code>	Flushes batch statements and commits database connection.
int	<code>delete(java.lang.String statement)</code>	Execute a delete statement.
int	<code>delete(java.lang.String statement, java.lang.Object parameter)</code>	Execute a delete statement.
<code>java.util.List&lt;BatchResult&gt;</code>	<code>flushStatements()</code>	Flushes batch statements.
<code>Configuration</code>	<code>getConfiguration()</code>	Retrieves current configuration.
<code>java.sql.Connection</code>	<code>getConnection()</code>	Retrieves inner database connection.
<code>&lt;T&gt; T</code>	<code>getMapper(java.lang.Class&lt;T&gt; type)</code>	Retrieves a mapper.
int	<code>insert(java.lang.String statement)</code>	Execute an insert statement.
int	<code>insert(java.lang.String statement, java.lang.Object parameter)</code>	Execute an insert statement with the given parameter object.
void	<code>rollback()</code>	Discards pending batch statements and rolls database connection back.
void	<code>rollback(boolean force)</code>	Discards pending batch statements and rolls database connection back.
void	<code>select(java.lang.String statement, java.lang.Object parameter, ResultHandler handler)</code>	Retrieve a single row mapped from the statement key and parameter using a ResultHandler.
void	<code>select(java.lang.String statement, java.lang.Object parameter, RowBounds rowBounds, ResultHandler handler)</code>	Retrieve a single row mapped from the statement key and parameter using a ResultHandler and RowBounds.
void	<code>select(java.lang.String statement, ResultHandler handler)</code>	Retrieve a single row mapped from the statement using a ResultHandler.
<code>&lt;T&gt; Cursor&lt;T&gt;</code>	<code>selectCursor(java.lang.String statement)</code>	A Cursor offers the same results as a List, except it fetches data lazily using an Iterator.
<code>&lt;T&gt; Cursor&lt;T&gt;</code>	<code>selectCursor(java.lang.String statement, java.lang.Object parameter)</code>	A Cursor offers the same results as a List, except it fetches data lazily using an Iterator.
<code>&lt;T&gt; Cursor&lt;T&gt;</code>	<code>selectCursor(java.lang.String statement, java.lang.Object parameter, RowBounds rowBounds)</code>	A Cursor offers the same results as a List, except it fetches data lazily using an Iterator.
<code>&lt;E&gt; java.util.List&lt;E&gt;</code>	<code>selectList(java.lang.String statement)</code>	Retrieve a list of mapped objects from the statement key.
<code>&lt;E&gt; java.util.List&lt;E&gt;</code>	<code>selectList(java.lang.String statement, java.lang.Object parameter)</code>	Retrieve a list of mapped objects from the statement key and parameter.
<code>&lt;E&gt; java.util.List&lt;E&gt;</code>	<code>selectList(java.lang.String statement, java.lang.Object parameter, RowBounds rowBounds)</code>	Retrieve a list of mapped objects from the statement key and parameter, within the specified row bounds.
<code>&lt;K,V&gt; java.util.Map&lt;K,V&gt;</code>	<code>selectMap(java.lang.String statement, java.lang.Object parameter, java.lang.String mapKey)</code>	The selectMap is a special case in that it is designed to convert a list of results into a Map based on one of the properties in the resulting objects.
<code>&lt;K,V&gt; java.util.Map&lt;K,V&gt;</code>	<code>selectMap(java.lang.String statement, java.lang.Object parameter, java.lang.String mapKey, RowBounds rowBounds)</code>	The selectMap is a special case in that it is designed to convert a list of results into a Map based on one of the properties in the resulting objects.
<code>&lt;K,V&gt; java.util.Map&lt;K,V&gt;</code>	<code>selectMap(java.lang.String statement, java.lang.String mapKey)</code>	The selectMap is a special case in that it is designed to convert a list of results into a Map based on one of the properties in the resulting objects.
<code>&lt;T&gt; T</code>	<code>selectOne(java.lang.String statement)</code>	Retrieve a single row mapped from the statement key.
<code>&lt;T&gt; T</code>	<code>selectOne(java.lang.String statement, java.lang.Object parameter)</code>	Retrieve a single row mapped from the statement key and parameter.
int	<code>update(java.lang.String statement)</code>	Execute an update statement.
int	<code>update(java.lang.String statement, java.lang.Object parameter)</code>	Execute an update statement.

## MyBatis 核心配置文件

核心配置文件包含了 MyBatis 最核心的设置和属性信息。如数据库的连接、事务、连接池信息等。

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!--MyBatis的DTD约束-->
3  <!DOCTYPE configuration
4      PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
5      "http://mybatis.org/dtd/mybatis-3-config.dtd">
6  <!--configuration 核心root标签-->
7  <configuration>
8      <!-- 属性可以在外部进行配置，并可以进行动态替换 -->
9      <properties resource="jdbc.properties" />
10
11     <settings>
12         <!-- 开启日志 -->
13         <setting name="logImpl" value="STDOUT_LOGGING" />
14         <!-- 开启下划线和驼峰式映射 -->
15         <setting name="mapUnderscoreToCamelCase" value="true"/>
16     </settings>
17
18     <!-- 配置别名 -->
19     <typeAliases>
20         <typeAlias type="com.rushuni.mybatis_demo.entity.Student"
21             alias="Student"/>
22     </typeAliases>
23
24     <!--environments 配置数据库环境，可以有多个。使用 default 属性指定使用的是哪个-->
25     <environments default="development">
26         <!--environment 配置数据库环境 id属性唯一标识-->
27         <environment id="development">
28             <!-- transactionManager 事务管理。 type属性，采用JDBC默认的事务-->
29             <transactionManager type="JDBC"/>
30             <!-- dataSource 数据源信息 type为 POOLED 为MyBatis默认连接池-->
31             <dataSource type="POOLED">
32                 <!-- property 连接数据库的配置信息 -->
33                 <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
34                 <property name="url" value="jdbc:mysql://localhost:3306/mybatis_demo?
35                     useUnicode=true&characterEncoding=UTF-8"/>
36                 <property name="username" value="${username}"/>
37                 <property name="password" value="${password}"/>
38             </dataSource>
39         </environment>
40     </environments>
41
42     <!-- mappers 引入映射配置文件 -->
43     <mappers>
44         <mapper resource="mapper/StudentMapper.xml"/>
45     </mappers>
46 </configuration>

```

## 事务

## 默认自动提交为 false

核心配置文件中设置：

```
<transactionManager type="JDBC"/>
```

默认 autoCommit 为false。

Shell

复制代码

```
1 Created connection 1985028494.
2 Setting autocommit to false on JDBC Connection
  [com.mysql.cj.jdbc.ConnectionImpl@7651218e]
3 ==> Preparing: insert into student (id,name,phone,age) values(?, ?, ?, ?)
```

此时，事务相关的 sql 语句都要显式调用 commit() 方法提交事务。

## 开启自动提交

通过 SqlSessionFactory 接口获取 SqlSession 对象时可以设置自动提交：

Java

复制代码

```
1 SqlSession openSession(boolean autoCommit);
```

# MyBatis 的 CRUD

## 插入记录

### 1. 修改 dao 层接口

修改接口文件，增加 insert 方法

Java

复制代码

```
1 /**
2     * 插入学生信息
3     * @param student
4     * @return
5     */
6 int insertStudent(Student student);
```

### 2. 实现 dao 层接口的方法

在接口的映射文件中，实现新增的方法。

注意 id 和方法名要一致。



```

1 <insert id="insertStudent">
2   insert into student (id,name,phone,age) values(#{id}, #{name}, #{phone}, #
   {age})
3 </insert>

```

### 3. 对新增的方法做单元测试

```

1 @Test
2 public void testInsert() throws IOException {
3     // 1. 核心配置文件名称
4     String config = "mybatis-config.xml";
5     // 2. 通过 MyBatis 提供的 Resource 类来读取核心配置文件
6     InputStream in = Resources.getResourceAsStream(config);
7     // 3. 获取 SqlSessionFactory 对象
8     SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);
9     // 4. 通过 SqlSessionFactory 对象获取 SqlSession 对象
10    SqlSession session = factory.openSession();
11    // 5. 构建要保存的对象
12    Student student = new Student();
13    student.setId(null);
14    student.setName("张二");
15    student.setPhone("13888888882");
16    student.setAge(22);
17    // 6. 通过 SqlSession 对象调用 dao 层接口中的方法。
18    int rows = session.getMapper(StudentMapper.class).insertStudent(student);
19    // 7. 事务提交
20    session.commit();
21    System.out.println("增加记录的行数:" + rows);
22    // 8. 关闭资源
23    session.close();
24 }

```

## 更新记录

### 1. 修改 dao 层接口

修改接口文件，增加 update 方法

```
1  /**
2   * 更新学生信息
3   * @param student
4   * @return
5   */
6  int updateStudent(Student student);
```

## 2. 实现 dao 层接口的方法

在接口的映射文件中，实现新增的方法。

注意 id 和方法名要一致。

```
1  <update id="updateStudent">
2    update student set age = #{age} where id=#{id}
3  </update>
```

## 3. 对方法做单元测试

```
1  @Test
2  public void testUpdate() throws IOException {
3      String config = "mybatis-config.xml";
4      InputStream in = Resources.getResourceAsStream(config);
5      SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);
6      SqlSession session = factory.openSession();
7      Student student = new Student();
8      student.setId(4);
9      student.setAge(23);
10     int rows = session.getMapper(StudentMapper.class).updateStudent(student);
11     session.commit();
12     System.out.println("更改记录的行数:"+rows);
13     session.close();
14 }
```

## 删除记录

### 1. 修改 dao 层接口

修改接口文件，增加 delete 方法

```

1  /**
2   * 删除学生信息
3   * @param id
4   * @return
5   */
6  int deleteStudent(int id);

```

## 2. 实现 dao 层接口的方法

在接口的映射文件中，实现新增的方法。

注意 id 和方法名要一致。

```

1  <delete id="deleteStudent">
2      delete from student where id=#{id}
3  </delete>

```

## 3. 对方法做单元测试

```

1  @Test
2  public void testDelete() throws IOException {
3      String config = "mybatis-config.xml";
4      InputStream in = Resources.getResourceAsStream(config);
5      SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);
6      SqlSession session = factory.openSession();
7      int id = 4;
8      int rows = session.getMapper(StudentMapper.class).deleteStudent(4);
9      session.commit();
10     System.out.println("删除记录的行数:"+rows);
11     session.close();
12 }

```

## 课堂作业

- 熟悉 MyBatis 使用流程，完成增删改查操作。

## MyBatis 工具类

创建 MybatisUtils 类把获取 sqlSession 对象的内容封装一下。

```
1 package com.rushuni.mybatis_demo.util;
2
3 import org.apache.ibatis.io.Resources;
4 import org.apache.ibatis.session.SqlSession;
5 import org.apache.ibatis.session.SqlSessionFactory;
6 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
7
8 import java.io.IOException;
9 import java.io.InputStream;
10
11 /**
12  * @author rushuni
13  * @date 2021年07月13日 2:59 下午
14  */
15 public class MybatisUtils {
16     static SqlSessionFactory factory;
17     static {
18         try (InputStream in = Resources.getResourceAsStream("mybatis-
19 config.xml")) {
20             factory = new SqlSessionFactoryBuilder().build(in);
21         } catch (IOException e) {
22             e.printStackTrace();
23         }
24     }
25     public static SqlSession getSqlSession() {
26         SqlSession sqlSession = null;
27         if (factory != null) {
28             sqlSession = factory.openSession();
29         }
30         return sqlSession;
31     }
32 }
```

## 课堂作业

- 熟悉 MyBatis 使用流程，使用工具类修改之前的增删改查操作。

## 传参


### 一个参数

要把参数传递到 mapper 映射文件中通常使用 `#{}`  这种方式。

dao 层接口：

```
1 Student getById(Integer id);
```


Java

 复制代码

dao 层实现：

```
1 <select id="getById" resultType="Student">
2     select id, name, phone, age from student where id = #{id}
3 </select>
```


XML

 复制代码

单元测试：

```
1 @Test
2 public void testGetById() throws IOException {
3     try (SqlSession session = MybatisUtils.getSqlSession()) {
4         StudentMapper mapper = session.getMapper(StudentMapperById.class);
5         Student student = mapper.getById(1);
6         System.out.println(student);
7     }
8 }
```

Java

 复制代码


## 多个参数

`#{value}` 是 MyBatis 的固定写法，但是 MyBatis 并没有提供多参数的固定写法，通常会设置 `parameterType` 为 `Map` 类型以实现多参数的传递。

dao 层接口：

```
1 /**
2     * 根据学生年龄查询学生信息
3     * @param ageMap
4     * @return
5     */
6 List<Student> listStudentWithAge(HashMap<String, Integer> ageMap);
```

Java

 复制代码

dao 层实现：

```

1 <select id="listStudentWithAge" resultType="Student">
2     select id, name, phone, age from student where age between #{min} and #
    {max};
3 </select>

```

单元测试：

```

1 @Test
2 public void testListStudentWithAge() throws IOException {
3     try (SqlSession session = MybatisUtils.getSqlSession()) {
4         StudentMapper mapper = session.getMapper(StudentMapper.class);
5         HashMap<String, Integer> stringIntegerHashMap = new HashMap<>();
6         stringIntegerHashMap.put("min", 18);
7         stringIntegerHashMap.put("max", 20);
8         List<Student> students =
9         mapper.listStudentWithAge(stringIntegerHashMap);
10        students.forEach(System.out::println);
11    }
12 }

```

## 课堂作业

- 给接口添加两个业务方法
  - 根据学生 id 查找学生信息
  - 根据学生的年龄查找学生信息

## 课后作业


1. 创建 maven 项目 mybatis-exam-01。
2. 添加 MyBatis 相关依赖。
3. 完成如下需求
  - a. 查询所有部门
  - b. 根据部门ID，查询某个部门的信息
  - c. 添加一个部门
  - d. 删除一个部门，根据部门 ID
  - e. 修改一个部门，根据部门 ID
  - f. 查询部门评分小于 60 分的所有部门信息
  - g. 查询部门人数小于 10 人且部门评分大于 80 分的所有部门信息

4. 对所有接口需求进行测试。
5. 少 copy。

部门表包含如下字段：

```
1  create table department (  
2      id int unsigned primary key auto_increment,  
3      name varchar(20),  
4      score varchar(20), -- 部门评分  
5      num int(3) -- 部门人数  
6  );
```

SQL

 复制代码