



服务计算与数据挖掘实验室

MyBatis开发技术

邹国兵，博士

副教授、博导/硕导

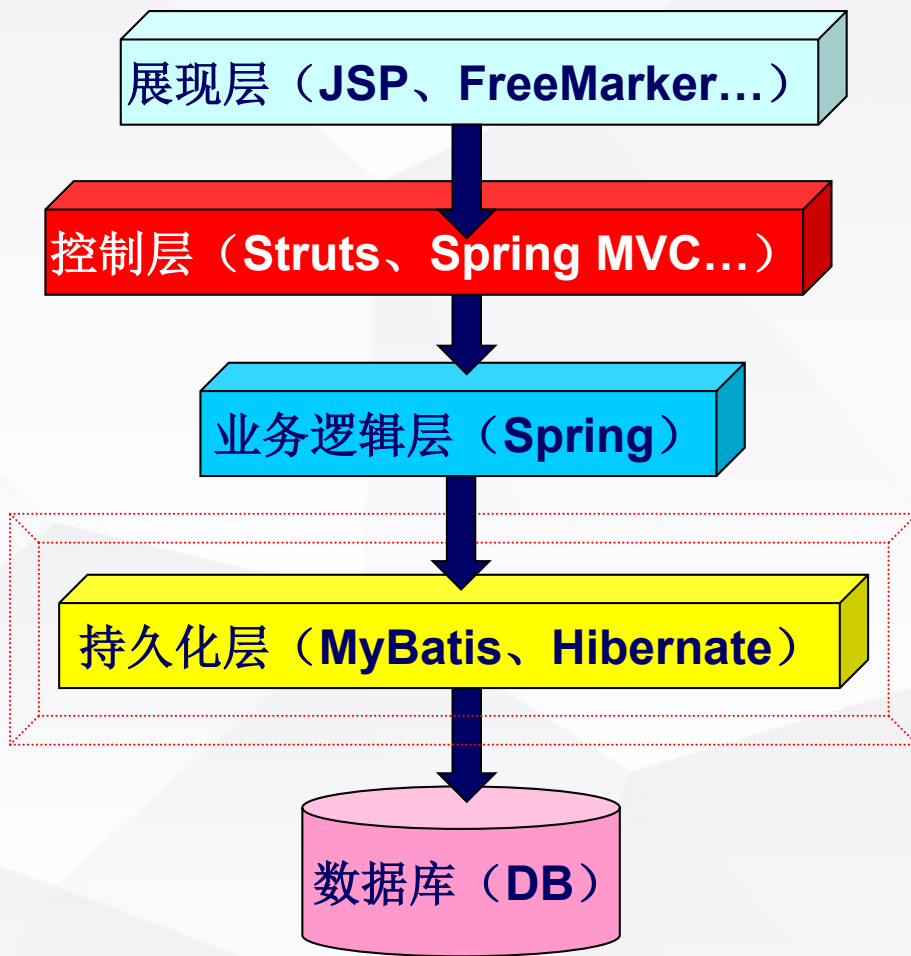
服务计算与数据挖掘实验室

<https://scdm-shu.github.io>

课程内容安排

- **ORM**框架背景
- **MyBatis**简介
- **MyBatis**体系结构
- **MyBatis**工作原理
- **MyBatis**核心配置
- **MyBatis**框架开发实例

ORM框架背景



Web应用的层次结构

ORM框架背景

- **Hibernate**是一个优秀的、标准的ORM框架，对数据库结构提供了较为**完整的封装**，提供了从**POJO**到数据库表的**全套**映射机制。
- 使用**Hibernate**持久化框架时，程序员只需定义好从**POJO**到数据库表的**映射关系**，即可通过**Hibernate**提供的类和接口完成持久化层的操作。
- 程序员甚至**不需要熟练掌握SQL语句**，**Hibernate**会根据定义的存储逻辑，自动生成对应的**SQL**并调用**JDBC**接口加以执行数据的持久化操作。

ORM框架背景

- 在大多数情况下（特别是对新项目、新系统的开发而言），**Hibernate**的这种机制无往不利，大有一统天下的势头。
- 但是，在一些**特定的环境**下，**Hibernate**的这种一站式解决方案却**未必适应**。

ORM框架背景

- 例如：

- 系统的部分或全部数据来自现有数据库，出于安全，只对开发团队提供几条 SQL语句（或存储过程），以获取数据，不公布具体的表结构。
- 系统数据处理量巨大，性能要求极为苛刻，这要求必须对SQL语句（或存储过程）进行高度优化。
- 开发规范中要求，所有涉及到业务逻辑的数据库操作，必须在数据库层由存储过程实现，如金融行业，商业银行在开发规范中严格指定。

ORM框架背景

- **Hibernate**提供了全面的数据库封装机制，相对于**MyBatis**可谓“**全自动化**”的**ORM**实现：实现了**POJO**和**数据库表**之间的映射，以及**SQL**的**自动生成和执行**。
- **Hibernate**数据持久化框架不适合解决上述问题，甚至无法使用它。
- 相对于**Hibernate**而言，“**半自动化**”的**MyBatis**正好可以解决上述问题。

ORM框架背景

- **MyBatis**提供的**ORM**机制，与**Hibernate**一样，为业务逻辑开发人员提供纯**Java**的持久化类。
- **MyBatis**的着力点在于**POJO/SQL**之间的映射关系。
- 具体讲，对于数据库操作，**Hibernate**自动生成**SQL**语句，而**MyBatis**并不为程序员在运行期间生成**SQL**语句，**具体的SQL语句需要程序员编写**，然后通过**映射**文件，将**SQL**所需的**参数及返回的结果字段**映射到指定的**POJO**。

ORM框架背景

- 相对Hibernate等“全自动”ORM框架而言，MyBatis以开发SQL的工作量和数据库移植性上的让步，为系统设计提供了更大的自由空间，作为“全自动”ORM实现的一种有益补充，MyBatis的存在具有特别的意义：
 - 可防止开发人员级的安全隐患；
 - 可进行更细致的SQL优化，可以减少查询字段；
 - 满足特定的开发规范要求，如用SQL语句或存储过程操作DB。

课程内容安排

- ORM框架背景
- **MyBatis**简介
- MyBatis体系结构
- MyBatis工作原理
- MyBatis核心配置
- MyBatis框架开发实例

MyBatis简介

- **MyBatis**的前身为**iBatis**，**iBatis**是由**Clinton Begin**开发，后来捐给**Apache**基金会，后者成立了**iBatis**开源项目。
- **iBatis**开源项目，**Apahce**基金会后来迁移到了**Google Code**，并且改名为**MyBatis**。

MyBatis简介

- **MyBatis**是当前主流的**Java**持久化框架之一，它与**Hibernate**一样，也是一种**ORM**框架。
- 因其性能优异，具有高度的灵活性、可优化性和易于维护等特点，所以受到广大互联网企业的青睐。
- **MyBatis**及其升级版（**Plus**）是目前大型互联网项目首选数据持久化框架。

MyBatis简介

- **MyBatis**是一个支持**普通SQL**查询、**存储过程**以及**高级映射**的持久化框架。
- 使用简单**XML**或注释进行**配置和原始映射**，用以将**Java**的**POJO**映射成数据库中的**记录**，使**Java**应用开发人员能够采用面向对象的编程思想操作数据库。

MyBatis简介

- **MyBatis**的优点:

1. 基于**SQL**语法，简单易学
2. 能了解底层组装过程
3. **SQL**语句封装在配置文件中，便于统一管理与维护，降低了程序的耦合度
4. 程序调试方便

MyBatis与传统JDBC比较

- 减少了大量的代码量（约**60%**）
- 架构级性能增强
- **SQL**代码从程序代码中彻底分离，可重用
- 增强了项目中的分工
- 增强了移植性

MyBatis与Hibernate的对比

- **MyBatis**是一个**SQL语句**映射的框架，相对于**Hibernate**全表映射是一个半自动化映射框架。
- 需要**手动匹配**提供**POJO**、**SQL**与映射关系，而**Hibernate**只需提供**POJO**和映射关系，不需要提供**SQL**。
- **手动写SQL语句**虽然比使用**Hibernate**的工作量大，但是**MyBatis**可配置动态的**SQL**并优化**SQL**，可通过**配置**决定**SQL**的映射规则，还支持**存储过程**。
- **MyBatis**更加适合**复杂和需要性能优化**的系统开发项目。

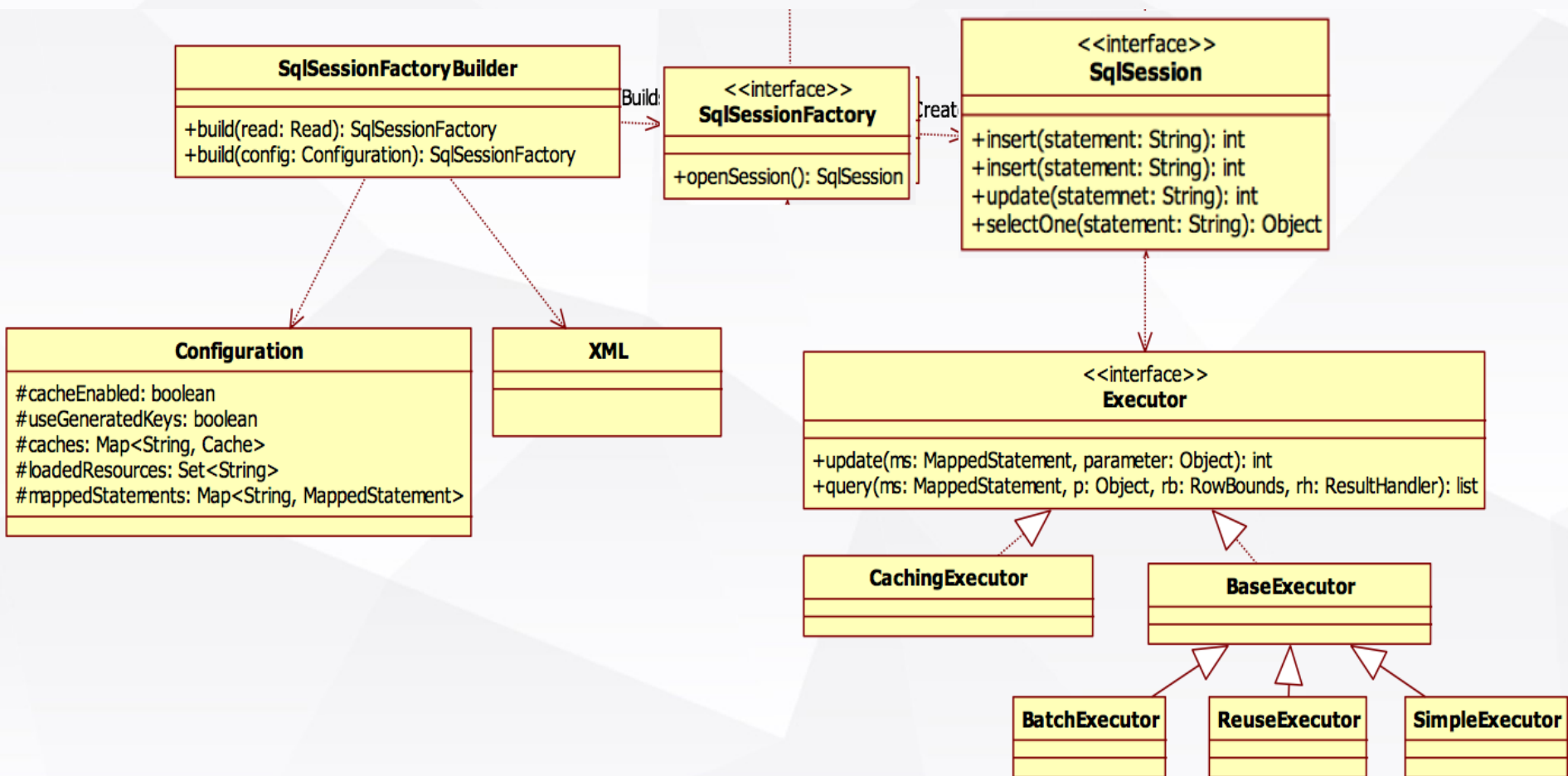
课程内容安排

- ORM框架背景
- MyBatis简介
- **MyBatis**体系结构
- MyBatis工作原理
- MyBatis核心配置
- MyBatis框架开发实例

MyBatis的体系结构

- **MyBatis**将开发人员从原始的**JDBC**访问中解放出来，开发人员只需要在**配置文件中**定义操作的**SQL语句**，无须关注底层**JDBC**操作，**以面向对象的方式进行持久化操作**。
- 底层数据库连接的获取、数据访问的实现、事务控制等等都由**MyBatis**框架完成。
- **MyBatis**中常用对象为**SqlSessionFactory**和**SqlSession**。

MyBatis核心接口和类的结构



SqlSessionFactory

什么是SqlSessionFactory?

- **SqlSessionFactory**是MyBatis框架中的一个主要对象，它是单个数据库映射关系经过编译后的内存镜像，其主要作用是创建**SqlSession**。
- **SqlSessionFactory**对象的实例由**SqlSessionFactoryBuilder**对象根据XML配置文件或预先定义的**Configuration**实例构建。

SqlSessionFactory

构建SqlSessionFactory

- 通过XML配置文件构建SqlSessionFactory实例的代码:

```
InputStream inputStream =  
Resources.getResourceAsStream(“配置文件位置”);  
  
SqlSessionFactory sqlSessionFactory = new  
SqlSessionFactoryBuilder().build(inputStream);
```

SessionFactory

- **SessionFactory**对象是线程安全的，它一旦被创建，在整个应用执行期间都会存在。
- 为节省资源，通常每一个数据库都会只对应一个**SessionFactory**，所以在构建**SessionFactory**实例时，通常使用单例模式。

SqlSession

什么是SqlSession?

- **SqlSession**是**MyBatis**框架中另一个重要的对象，它是应用程序与持久层之间进行交互操作的一个单线程对象，其主要作用是执行持久化操作。
- **SqlSession**实例是线程不安全的，是不能被共享的，每一个线程都应该有一个**SqlSession**实例。

SqlSession

如何使用SqlSession?

- 调用**selectOne**, **selectList**, **insert**, **update**, **delete**等方法执行增删改查等操作。

SqlSession

SqlSession中的方法

查询方法:

- `<T> T selectOne(String statement);` //返回一条泛型对象
其中的statement通常是Mapper中定义的SQL语句id。
- `<T> T selectOne(String statement, Object parameter);`
- `<E> List<E> selectList(String statement);` //返回泛型对象集合
- `<E> List<E> selectList(String statement, Object parameter);`
- `<E> List<E> selectList(String statement, Object parameter, RowBounds rowBounds);` //用于分页的参数对象

SqlSession

SqlSession中的方法

插入、更新和删除方法：

- `int insert(String statement);`
- `int insert(String statement, Object parameter);`
- `int update(String statement);`
- `int update(String statement, Object parameter);`
- `int delete(String statement);`
- `int delete(String statement, Object parameter);`

SqlSession

SqlSession中的方法

其它方法：

- **void commit();** 提交事务的方法。
- **void rollback();** 回滚事务的方法。
- **void close();** 关闭SqlSession对象。
- **Connection getConnection();** 获取JDBC数据库连接对象。

SqlSession

- SqlSession的使用

```
public class MybatisUtils {  
    private static SqlSessionFactory sqlSessionFactory = null;  
    static {  
        try { //只读文本为AsReader  
            Reader reader = Resources.getResourceAsReader("mybatis-  
                config.xml");  
            sqlSessionFactory = new  
                SqlSessionFactoryBuilder().build(reader);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
    public static SqlSession getSession() {  
        return sqlSessionFactory.openSession();  
    }  
}
```

SqlSession

- 使用完**SqlSession**对象后要及时关闭，通常可以将其放在**finally**块中关闭。

```
SqlSession sqlSession
```

```
    = MybatisUtils.getSession();
```

```
try {
```

```
    // 此处执行持久化操作
```

```
    sqlSession.insert()/update()/delete()/selectOne()/selectList();
```

```
} finally {
```

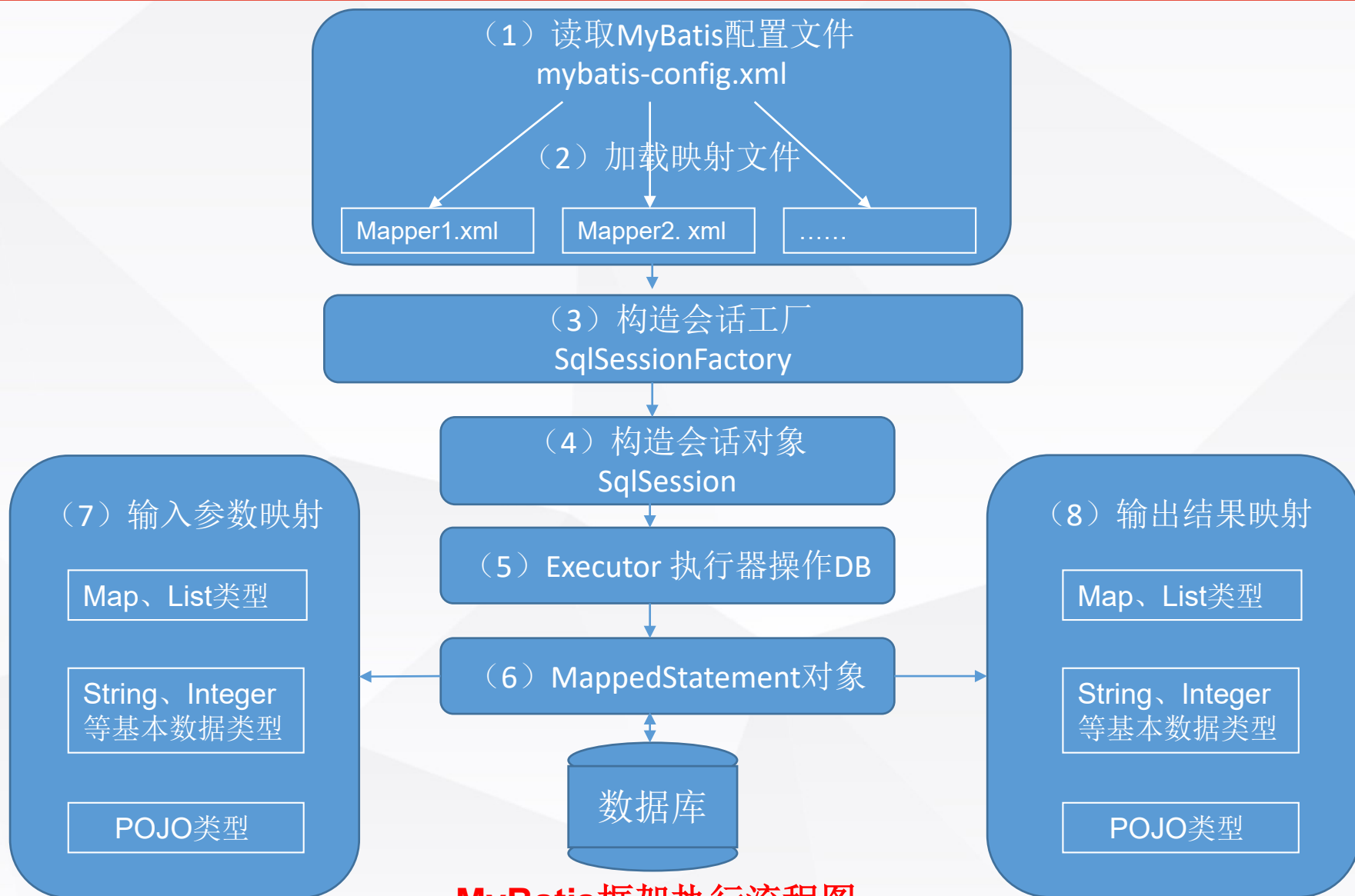
```
    sqlSession.close();
```

```
}
```

课程内容安排

- ORM框架背景
- MyBatis简介
- MyBatis体系结构
- **MyBatis工作原理**
- MyBatis核心配置
- MyBatis框架开发实例

MyBatis工作流程



MyBatis框架执行流程图

MyBatis工作流程

MyBatis框架在操作数据库时，经历以下8个步骤：

（1）读取**mybatis**配置文件

- **mybatis-config.xml**作为**MyBatis**的全局配置文件，配置了**MyBatis**的运行环境等信息，包括数据库的连接信息。

（2）加载映射文件

- 映射文件**mapper.xml**文件（**SQL**映射文件）配置了操作数据库的**SQL**语句，每个映射文件对应数据库中的一张表，文件需要在**mybatis-config.xml**中加载才能起作用。

MyBatis工作流程

(3) 构造会话工厂

- 通过**Mybatis**的环境等配置信息构造**SqlSessionFactory**。

(4) 创建会话对象

- 由会话工厂创建**SqlSession**对象，该对象包含了执行**SQL**语句的所有方法，操作数据库需要通过**SqlSession**执行。

MyBatis工作流程

(5) MyBatis的Executor执行器接口操作DB

- 通过MyBatis底层自定义的Executor执行器接口来操作数据库。
- Executor执行器接口具有两个实现：
基本执行器（Base Executor）
缓存执行器（Caching Executor）。
- Executor执行器接口根据SqlSession对象传递的参数动态地生成需要执行的SQL语句。

MyBatis工作流程

(6) 映射的语句对象传递给Executor接口的执行方法

- **mapper.xml**文件中的一个**SQL**语句对应**MyBatis**的一个底层对象即**MappedStatement**语句对象，它作为一个参数传递给**Executor**接口的执行方法。
- **MappedStatement**是对**映射信息**的封装，存储要映射的**SQL**语句的**ID**、**参数**等。

MyBatis工作流程

(7) 输入参数映射

- 在执行**Executor**接口的方法时，**MappedStatement**对象会对用户要执行的**SQL**语句的**输入参数**进行定义（可定义为**HashMap**、**List**类型、基本类型和**POJO**类型）。
- **Executor**通过**MappedStatement**在执行**SQL**前将输入的**Java**对象映射至**SQL**中。
- **输入参数映射**就是**JDBC**编程中对**PreparedStatement**设置参数。

MyBatis工作流程

(8) 输出结果映射

- 在数据库中执行完SQL语句之后，MappedStatement对象会对SQL的执行输出结果进行定义（定义为HashMap、List、基本类型、POJO类型）。
- Executor执行器通过MappedStatement在执行SQL语句后，将输出结果映射至Java对象中。
- 输出结果映射过程相当于JDBC编程中对结果的解析处理过程。

课程内容安排

- ORM框架背景
- MyBatis简介
- MyBatis体系结构
- MyBatis工作原理
- **MyBatis核心配置**
- MyBatis框架开发实例

基础配置文件mybatis-config.xml

mybatis-config.xml是系统的核心配置文件，包含数据源和事务管理等设置和属性信息，**XML**文档结构如下：

configuration 配置文件根元素

properties 可以配置Java 属性在配置文件中

settings 修改 **MyBatis** 在运行时的行为方式

typeAliases 为 **Java** 类型命名一个短的名字

typeHandlers 类型处理器

objectFactory 对象工厂

plugins 插件

environments 环境

environment 环境变量

transactionManager 事务管理器

dataSource 数据源

mappers 映射器

<properties> 元素

- 通过配置属性元素，使用外部的配置来动态替换内部定义的属性
如在src目录下，添加一个全名为db.properties的File,代码如下：

```
jdbc.driver=com.mysql.jdbc.Driver
```

```
jdbc.url=jdbc:mysql://localhost:3306/mybatis
```

```
jdbc.username=root
```

```
jdbc.password=admin
```

- 在mybatis-config.xml中配置<properties..../> 属性:

```
<properties resource="db.properties" />
```


settings设置元素< settings >

- **setting**是**MyBatis**框架中的调整设置，它们会改变**MyBatis**的运行时的一些行为。例如，**缓存、延迟加载、结果集控制、执行器、分页设置、命名规则**等一系列控制性参数，其所有的 **setting** 配置都放在父标签 **settings** 标签中。
- 一个配置完整的 **settings** 元素的示例如下：

settings设置元素< settings >

<settings>


1. <setting name="cacheEnabled" value="true"/>
 2. <setting name="lazyLoadingEnabled" value="true"/>
 3. <setting name="multipleResultSetsEnabled" value="true"/>
 4. <setting name="useColumnLabel" value="true"/>
 5. <setting name="useGeneratedKeys" value="false"/>
 6. <setting name="autoMappingBehavior" value="PARTIAL"/>
 7. <setting name="autoMappingUnknownColumnBehavior" value="WARNING"/>
 8. <setting name="defaultExecutorType" value="SIMPLE"/>
 9. <setting name="defaultStatementTimeout" value="25"/>
 10. <setting name="defaultFetchSize" value="100"/>
 11. <setting name="safeRowBoundsEnabled" value="false"/>
 12. <setting name="mapUnderscoreToCamelCase" value="false"/>
 13. <setting name="localCacheScope" value="SESSION"/>
 14. <setting name="jdbcTypeForNull" value="OTHER"/>
 15. <setting name="lazyLoadTriggerMethods"
value="equals,clone,hashCode,toString"/>
- </settings>

<typeAliases> 元素

- 为配置文件中java类型设置别名.

<!-- 定义别名 -->

<typeAlias alias="user" type="com.mybatis.po.User" />



This screenshot shows the configuration of the <typeAliases> element in the mybatis-config.xml file. The file is opened in an IDE with tabs for UserMapper.xml, UserDaoTest.java, and *mybatis-config.xml. The configuration includes a comment explaining the purpose of the element and a specific <typeAlias> tag. A red box highlights the alias value "User" in the tag.

```
12 </settings>
13
14 <typeAliases>
15   <!-- 为一个pojo类去指定一个简称。然后在mapper.xml中使用的时候，可以直接书写简写就可以
16   type: 指定类的全路径，alias : 别名，简写 -->
17   <typeAlias type="cn.yanqi.mybaitis.pojo.User" alias="User"/>
18
19 </typeAliases>
20
```



This screenshot shows the configuration of the <select> element in the UserMapper.xml file. The file is opened in an IDE with tabs for UserMapper.xml, UserDaoTest.java, and mybatis-config.xml. The configuration includes the XML declaration, DOCTYPE, and the <select> tag. A red box highlights the resultType value "User" in the tag. A red arrow points from the "User" alias in the mybatis-config.xml file to this "User" value, demonstrating how the alias is used.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6 <mapper namespace="cn.yanqi.mybaitis.dao.UserDao">
7   <!-- 根据id来查询 -->
8   <select id="queryUserById" resultType="User">
9     select * from t_user where id = #{id}
10  </select>
11
```

<typeAliases> 元素

- 当pojo类过多时,还可以配置扫描包形式自定义别名

<typeAliases>

<package name="com.mybatis.po" />

</typeAliases>

```
<typeAliases>
  <!-- 为一个pojo类去指定一个简称。然后在mapper.xml中使用的时候，可以直接书写简写就可以
  type: 指定类的全路径， alias : 别名，简写 -->
  <!-- <typeAlias type="cn.yanqi.mybatis.pojo.User" alias="User"/> -->

  <!-- 把指定的包下面所有类，都去起别名。
       简称默认就是类的名字，并且不区分大小写。建议大家，直接使用类的名字
  -->
  <package name="cn.yanqi.mybatis.pojo"/>
</typeAliases>
```

https://blog.csdn.net/Denial_learn

```
<!-- 查询所有 -->
<select id="queryAllUser" resultType="User">
  Select * from t_user;
</select>
```

<environments>元素- 基础配置

在配置文件中修改数据库连接的信息:

<!--1.配置环境，默认的环境id为mysql -->

<environments default="mysql">

<!--1.2.配置id为mysql的数据库环境 -->

<environment id="mysql">

<!-- 使用JDBC的事务管理 -->

<transactionManager type="JDBC" />

<!--数据库连接池 -->

<dataSource type="POOLED">

<!-- 数据库驱动 -->

<property name="driver" value="{jdbc.driver}" />

<!-- 连接数据库的url -->

<property name="url" value="{jdbc.url}" />

<!-- 连接数据库的用户名 -->

<property name="username" value="{jdbc.username}" />

<!-- 连接数据库的密码 -->

<property name="password" value="{jdbc.password}" />

</dataSource>

</environment>

</environments>

<environments>元素—事务管理

MyBatis 有两种事务管理类型

- **JDBC** - 这个类型全部使用 **JDBC** 的提交和回滚功能。它依靠使用连接的数据源来管理事务的作用域。
- **MANAGED** - 这个类型什么不做，它从不提交、回滚和关闭连接，而是让容器来管理事务的全部生命周期。

<environments>元素—数据源

数据源类型有三种： **UNPOOLED** ， **POOLED** ， **JNDI**

- **UNPOOLED** - 这个数据源实现只是在每次请求的时候简单地打开和关闭一个连接，效率较低；对不需要性能和立即响应的简单应用来说，它是一种合适选择。
- **POOLED** - 这个数据源缓存 **JDBC** 连接对象，用于避免每次都要连接和生成连接实例而需要的验证时间。对于并发 **Web** 应用，这种方式非常流行，因为它有最快的响应时间。
- **JNDI** - 这个数据源实现是为了准备和 **Spring** 或应用服务一起使用，可以在外部也可以在内部配置这个数据源，然后在 **JNDI** 上下文中引用它。通常需要上下文环境配置文件，提供数据库链接信息。

配置中的SQL映射文件

SQL映射文件

//配置Mapper的位置，使用相对路径

```
<mappers>  
  <mapper resource="mapper/Category.xml"/>  
</mappers>
```


SQL映射文件

SQL 映射文件主要语句：

select – 映射查询语句

insert – 映射插入语句

update – 映射更新语句

delete – 映射删除语句

SQL映射文件 - Select标签

```
<!-- namespace表示命名空间 -->
<mapper namespace="com.mybatis.mapper.CustomerMapper">
    <!--根据客户编号获取客户信息 -->
    <select id="findCustomerById" parameterType="Integer"
            resultType="com.mybatis.po.Customer">
        select * from t_customer where id = #{id}
    </select>
```

SQL映射文件 - Select标签

调用**Select**语句查询客户信息举例：

```
@Test
public void findCustomerByIdTest() throws Exception {
    // 1、读取配置文件
    String resource = "mybatis-config.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    // 2、根据配置文件构建SqlSessionFactory
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    // 3、通过SqlSessionFactory创建SqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 4、SqlSession执行映射文件中定义的SQL，并返回映射结果
    Customer customer = sqlSession.selectOne("com.mybatis.mapper.CustomerMapper.findCustomerById", 1);
    // 打印输出结果
    System.out.println(customer.toString());
    // 5、关闭SqlSession
    sqlSession.close();
}
```

SQL映射文件 - Insert标签

```
<!-- 添加客户信息 -->  
<insert id="addCustomer" parameterType="com.mybatis.po.Customer">  
    insert into t_customer(username,jobs,phone)  
    values("#{username"},#{jobs},#{phone})  
</insert>
```

SQL映射文件 - Insert标签

```
/**
 * 添加客户
 */
@Test
public void addCustomerTest() throws Exception {
    // 1、读取配置文件
    String resource = "mybatis-config.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    // 2、根据配置文件构建SqlSessionFactory
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    // 3、通过SqlSessionFactory创建SqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 4、SqlSession执行添加操作
    // 4.1创建Customer对象，并向对象中添加数据
    Customer customer = new Customer();
    customer.setUsername("rose");
    customer.setJobs("student");
    customer.setPhone("13333533092");
    // 4.2执行SqlSession的插入方法，返回的是SQL语句影响的行数
    int rows = sqlSession.insert("com.mybatis.mapper.CustomerMapper.addCustomer", customer);
    // 4.3通过返回结果判断插入操作是否执行成功
    if (rows > 0) {
        System.out.println("您成功插入了" + rows + "条数据！");
    } else {
        System.out.println("执行插入操作失败！！！");
    }
    // 4.4提交事务
    sqlSession.commit();
    // 5、关闭SqlSession
    sqlSession.close();
}
```

SQL映射文件 - Update标签和Delete标签

- Update

```
<!-- 更新客户信息 -->  
<update id="updateCustomer" parameterType="com.mybatis.po.Customer">  
    update t_customer set  
        username=#{username}, jobs=#{jobs}, phone=#{phone}  
    where id=#{id}  
</update>
```

- Delete

```
<!-- 删除客户信息 -->  
<delete id="deleteCustomer" parameterType="Integer">  
    delete from t_customer where id=#{id}  
</delete>
```

课程内容安排

- ORM框架背景
- MyBatis简介
- MyBatis体系结构
- MyBatis工作原理
- MyBatis核心配置
- **MyBatis框架开发实例**

MyBatis框架开发实例

使用**MyBatis**框架开发应用的过程如下：

（1）准备数据库和数据库连接

```
CREATE TABLE t_customer  
(  
id int identify(1, 1) not null primary key,  
username varchar(50) ,  
jobs varchar(16),  
phone varchar(15)  
);
```


MyBatis框架开发实例

(2) 创建持久化类; // Customer.java

```
package com.shu.ces.pojo;
```

```
public class Customer {
```

```
    private Integer id;    // 主键id
```

```
    private String username; // 客户名称
```

```
    private String jobs;    // 职业
```

```
    private String phone;   // 电话
```

```
    public Integer getId() { return id; }
```

```
    public void setId(Integer id) { this.id = id; }
```

```
    public String getUsername() { return username; }
```

```
    public void setUsername(String username) { this.username = username; }
```

```
    public String getJobs() { return jobs; }
```

```
    public void setJobs(String jobs) { this.jobs = jobs; }
```

```
    public String getPhone() { return phone; }
```

```
    public void setPhone(String phone) { this.phone = phone; }
```

```
    @Override
```

```
    public String toString() {
```

```
        return "Customer [id=" + id + ", username=" + username +
```

```
            ", jobs=" + jobs + ", phone=" + phone + "];"
```

```
    }
```

```
}
```

MyBatis框架开发实例

(3) 构建配置文件;

//mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE configuration
```

```
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
```

```
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
```

```
<configuration>
```

```
    <typeAliases>
```

```
        <package name="com.shu.ces.pojo"/>
```

```
    </typeAliases>
```

MyBatis框架开发实例

```
<environments default="development">
  <environment id="development">
    <transactionManager type="JDBC"/>
    <dataSource type= "POOLED" >
      <property name= "driver" value= "com.mysql.jdbc.Driver" />
      <property name= "url" value= "jdbc:mysql://localhost:3306/test" />
      <property name= "username" value= "root" />
      <property name="password" value="123456"/>
    </dataSource>
  </environment>
</environments>
<mappers>
  <mapper resource= "com/shu/ces/mapper/CustomerMapper.xml"/>
</mappers>
</configuration>
```

MyBatis框架开发实例

(3) 构建映射文件；SQL（非表）与POJO之间的映射

// **CustomerMapper.xml**

```
<?xml version= "1.0" encoding= "UTF-8" ?>
```

```
<!DOCTYPE mapper
```

```
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
```

```
    http://mybatis.org/dtd/mybatis-3-mapper.dtd>
```

```
<mapper namespace="com.shu.ces.mapper.CustomerMapper">
```

```
    <!--根据客户编号获取客户信息 -->
```

```
        <select id="findCustomerById" parameterType="Integer"
            resultType="com.shu.ces.pojo.Customer">
            select * from t_customer where id = #{id}
```

```
        </select>
```

```
    <!--根据客户名模糊查询客户信息列表-->
```

```
        <select id="findCustomerByName" parameterType="String"
            resultType=" com.shu.ces.pojo.Customer">
            select * from t_customer where username like
            concat('%',#{value},'%')
```

```
        </select>
```

MyBatis框架开发实例

<!-- 添加客户信息 -->

```
<insert id="addCustomer" parameterType="com.shu.ces.pojo.Customer">  
    insert into t_customer(username, jobs, phone)  
    values("#{username}", #{jobs}, #{phone})  
</insert>
```

<!-- 更新客户信息 -->

```
<update id="updateCustomer" parameterType="com.shu.ces.pojo.Customer">  
    update t_customer set  
    username=#{username}, jobs=#{jobs}, phone=#{phone}  
    where id=#{id}  
</update>
```

<!-- 删除客户信息 -->

```
<delete id="deleteCustomer" parameterType="Integer">  
    delete from t_customer where id=#{id}  
</delete>
```

```
</mapper>
```

MyBatis框架开发实例

(4) 应用测试类

// MybatisTest.java

package com.shu.ces.test;

import com.shu.ces.pojo.Customer;

import java.io.InputStream;

import java.util.List;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

MyBatis框架开发实例

```
public class TestMybatis {  
    public static void main(String[] args) throws IOException {  
        String resource = "mybatis-config.xml";  
        InputStream inputStream =  
            Resources.getResourceAsStream(resource);  
        SqlSessionFactory sqlSessionFactory = new  
            SqlSessionFactoryBuilder().build(inputStream);  
        SqlSession session=session.openSession();  
        Customer c=session.selectOne("com.shu.ces.mapper"+"  
            findCustomerById");  
        for (Customer n :c ) {  
            System.out.println(n.getUserName());  
        }  
    }  
}
```

MyBatis框架开发实例

**IntelliJ IDEA演示
(增加一个产品实例)**