# Mybatis自定义框架及使用

## 文档结构

### 1.1目录视图

1. Mybatis入门

·mybatis的概述

·mybatis的环境搭建

·mybatis入门案例

1. mybatis基本使用

·mybatis的单表curd操作

·mybatis的参数和返回值

·mybatis的dao编写

·mybatis的配置细节

几个标签的使用

1. mybatis的深入和多表

·mybatis的连接池

·mybatis的事务控制及设计的方法

·mybatis的多表查询

一对多（多对一）

多对多

1. mybatis的缓存和注解开发

·mybatis中的加载时机（查询的时机）

·mybatis的一级缓存和二级缓存

·mybatis的注解开发

单表CURD

多表查询

## 第2节 框架概述

### 2.1什么是框架

（1）什么是框架

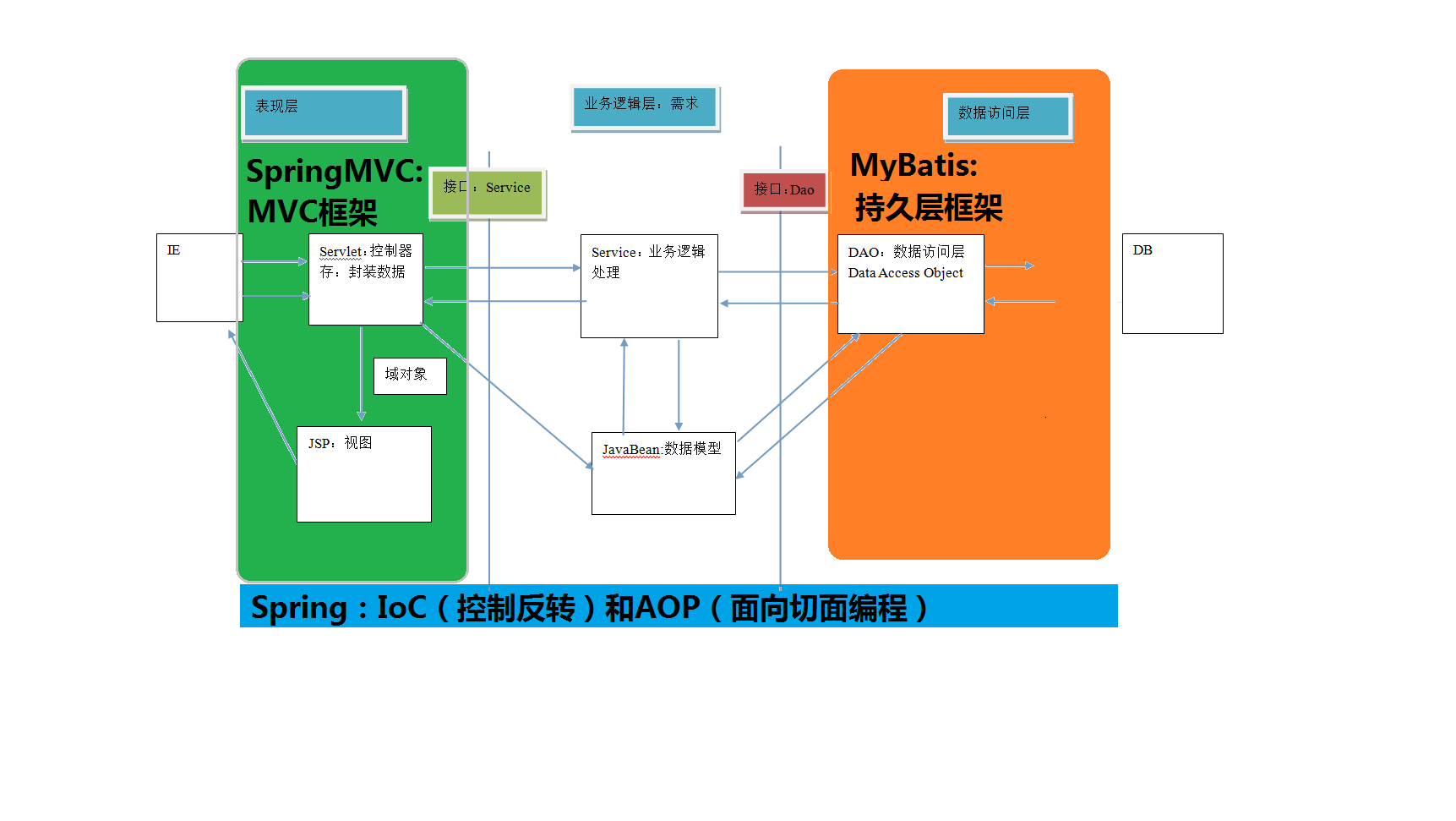
框架（Framework）是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法;另一种定义认为，框架是可被应用开发者定制的应用骨架。前者是从应用方面而后者是从目的方面给出的定义简而言之，框架其实就是某种应用的半成品，就是一组组件，供你选用完成你自己的系统。简单说就是使用别人搭好的舞台，你来做表演。而且，框架一般是成熟的，不断升级的软件。

他是我们软件开发中的一套解决方案，不同框架解决不同的问题。

使用框架的好处：

框架封装了很多细节，使开发者能够提高开发效率，使用极简的方式实现功能。大大提高开发效率

1. 框架要解决的问题



·应用自身的设计与具体实现解耦；

·框架一般处在底层应用平台（如J2EE）和高层业务逻辑之间的中间层

（3）软件开发分层

·表现层：

用于展示数据

·业务层：

处理业务需求

·持久层

和数据库交互

（4）持久层技术解决方案

JDBC技术：

Connection

PreparedStatement

ResultStatement

String的JdbcTemplate：

String对JDBC的简单封装

Apache的DBUtils：

他和Spring的JdbcTemplate很像，也是对Jdbc的简单封装

以上都不是框架

JDBC是规范

Spring的JdbcTemplate和Apache的DBUtils都只是工具类

1. MyBatis概述

基于Java的持久层框架，内部封装了jdbc，使开发者需要关注sql语句半数，而不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。

注解

XML

ORM：Object Relational Mapping对象关系映射

简单的说：

就是把数据库表和实体类及实体类的属性对应起来

让我们可以操作实体类就实现数据库操作

Uase\_table User

id userId

User\_name userName

### 2.1JMybatis的入门

1. Mybatis的环境搭建

第一步：创建maven工程并导入坐标

第二步：创建实体类和dao的接口

第三步：创建mybatis的主配置文件

SqlMapConfig.xml

第四步：创建映射配置文件

IUserDao.xml

环境搭建的注意事项：

第一个：创建IUserDao.xml和IUserDao.java时，名称是为了和我们以前的知识保持一致。在Mybatis中它把持久层的操作接口名称和映射文件也叫作Mapper

所以：IUserDao和IUserMapper是一样的

第二个：在idea中创建目录的时候，它和包是不一样的

包在创建时：com.byf.dao他是三级结构

目录在创建时 com/byf/dao是三级目录，但com.byf.dao是一级目录

第三个：mybatis的映射配置文件必须和dao接口包结构相同

第四个：映射配置文件的mapper标签namespace属性的取值必须是dao接口的全限定类名

第五个：映射配置文件的操作配置（select），id属性的取值必须是dao接口的方法名。

当我们遵从了三四五点之后，我们在开发中就无需再写dao的实现类

### 2.3环境搭建

|  |
| --- |
| @ToString **public class** User **implements** Serializable {  **private** Integer **id**;  **private** String **username**;  **private** Date **birthday**;  **private** String **sex**;  **private** String **address**;   **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** Date getBirthday() {  **return** (Date) **birthday**.clone();  }   **public void** setBirthday(Date birthday) {  **this**.**birthday** = birthday;  }   **public** String getSex() {  **return sex**;  }   **public void** setSex(String sex) {  **this**.**sex** = sex;  }   **public** String getAddress() {  **return address**;  }   **public void** setAddress(String address) {  **this**.**address** = address;  } } |
| **public interface** IUserDao {  */\*\*  \* 查询所有操作  \** ***@return*** *\*/* List<User> findAll(); } |

G:\Mybatis\mybatis01\src\main\resources\SqlMapConfig.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**>  *<!-- 配置环境 -->* <**environments default="mysql"**>  *<!-- 配置mysql环境 -->* <**environment id="mysql"**>  *<!-- 配置事务类型 -->* <**transactionManager type="JDBC"**></**transactionManager**>  *<!-- 配置数据源（连接池） -->* <**dataSource type="POOLED"**>  <**property name="driver" value="com.mysql.jdbc.Driver"**/>  <**property name="url" value="jdbc:mysql://localhost:3306/eesy\_mybatis"**/>  <**property name="username" value="root"**/>  <**property name="password" value="1234"**/>  </**dataSource**>  </**environment**>  </**environments**>   *<!--指定映射配置文件的位置，映射配置文件指的是每个dao独立的配置文件 -->* <**mappers**>  <**mapper resource="com/byf/dao/IUserDao.xml"**></**mapper**>  </**mappers**> </**configuration**> |

G:\Mybatis\mybatis01\src\main\resources\com\byf\dao\IUserDao.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.byf.dao.IUserDao"**>  *<!-- 配置查询所有 -->* <**select id="findAll"**>  select *\** from user  </**select**>  </**mapper**> |

G:\Mybatis\mybatis01\src\main\resources\log4j.properties

|  |
| --- |
| *# Set root category priority to INFO and its only appender to CONSOLE. #log4j.rootCategory=INFO, CONSOLE debug info warn error fatal* **log4j.rootCategory**=**debug, CONSOLE, LOGFILE** *# Set the enterprise logger category to FATAL and its only appender to CONSOLE.* **log4j.logger.org.apache.axis.enterprise**=**FATAL, CONSOLE** *# CONSOLE is set to be a ConsoleAppender using a PatternLayout.* **log4j.appender.CONSOLE**=**org.apache.log4j.ConsoleAppender log4j.appender.CONSOLE.layout**=**org.apache.log4j.PatternLayout log4j.appender.CONSOLE.layout.ConversionPattern**=**%d{ISO8601} %-6r [%15.15t] %-5p %30.30c %x - %m\n** *# LOGFILE is set to be a File appender using a PatternLayout.* **log4j.appender.LOGFILE**=**org.apache.log4j.FileAppender log4j.appender.LOGFILE.File**=**d:**\a**xis.log log4j.appender.LOGFILE.Append**=**true log4j.appender.LOGFILE.layout**=**org.apache.log4j.PatternLayout log4j.appender.LOGFILE.layout.ConversionPattern**=**%d{ISO8601} %-6r [%15.15t] %-5p %30.30c %x - %m\n** |

|  |
| --- |
| **public class** MybatisTest {  @Test  **public void** testConfig() **throws** IOException {  *// 1.读取配置文件* InputStream in = Resources.*getResourceAsStream*(**"SqlMapConfig.xml"**);  *// 2.创建SqlSessionFactory工厂* SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  SqlSessionFactory factory = builder.build(in);  *//3.使用工厂生产生产SqlSession对象* SqlSession session = factory.openSession();  *//4.使用SqlSession创建Dao接口的代理对象* IUserDao userDao = session.getMapper(IUserDao.**class**);  *//5.使用代理对象执行方法* List<User> users = userDao.findAll();  **for** (User user:users){  System.***out***.println(user);  }  *//6.释放资源* session.close();  } } |
| **[ main] DEBUG ansaction.jdbc.JdbcTransaction - Opening JDBC Connection**  **[ main] DEBUG source.pooled.PooledDataSource - Created connection 422392391.**  **[ main] DEBUG ansaction.jdbc.JdbcTransaction - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@192d3247]**  **[ main] DEBUG com.byf.dao.IUserDao.findAll - ==> Preparing: select \* from user**  **[ main] DEBUG com.byf.dao.IUserDao.findAll - ==> Parameters:**  **[ main] DEBUG com.byf.dao.IUserDao.findAll - <== Total: 6**  **User(id=41, username=Jack, birthday=Tue Feb 27 17:47:08 CST 2018, sex=男, address=北京)**  **User(id=42, username=Tom, birthday=Fri Mar 02 15:09:37 CST 2018, sex=女, address=北京金燕龙)**  **User(id=43, username=Jerry, birthday=Sun Mar 04 11:34:34 CST 2018, sex=女, address=北京金燕龙)**  **User(id=45, username=Bob, birthday=Sun Mar 04 12:04:06 CST 2018, sex=男, address=北京金燕龙)**  **User(id=46, username=Alice, birthday=Wed Mar 07 17:37:26 CST 2018, sex=男, address=北京)**  **User(id=48, username=John, birthday=Thu Mar 08 11:44:00 CST 2018, sex=女, address=北京修正)**  **[ main] DEBUG ansaction.jdbc.JdbcTransaction - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection@192d3247]**  **[ main] DEBUG ansaction.jdbc.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@192d3247]**  **[ main] DEBUG source.pooled.PooledDataSource - Returned connection 422392391 to pool.** |

### 2.4Mybatis中的设计模式

为什么在拿到代理对象执行方法前，要增加中间的创建类：

为了能够利用重载实现更灵活的配置加载

|  |
| --- |
| **public class** MybatisTest {  @Test  **public void** testConfig() **throws** IOException {  */\*\*  \* 1.读取配置文件  \* 1）绝对路径：d:/xxx/xxx.xml 不可用  \* 2）相对路径：src/java/main/xxx.xml 不可用  \* 3）使用类加载器，他只能兑取类路径的配置文件  \* 4）使用ServletContext对象的getRealPath()  \*/* InputStream in = Resources.*getResourceAsStream*(**"SqlMapConfig.xml"**);   */\*\*  \* 2.创建SqlSessionFactory工厂  \* 1）创建工厂mybatis使用了构建者模式  \* 优势：把对象创建的细节。因此，使用者直接调用对象  \*/* SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  SqlSessionFactory factory = builder.build(in);   */\*\*  \* 3.使用工厂生产生产SqlSession对象  \* 1）生成SqlSession使用了工厂模式  \* 优势：解耦（降低类之间的依赖关系）  \*/* SqlSession session = factory.openSession();   */\*\*  \* 4.使用SqlSession创建Dao接口的代理对象  \* 1）创建Dao接口实现类使用了代理模式  \* 优势：不修改源码的基础上，增强方法  \*/* IUserDao userDao = session.getMapper(IUserDao.**class**);  *//5.使用代理对象执行方法* List<User> users = userDao.findAll();  **for** (User user:users){  System.***out***.println(user);  }  *//6.释放资源* session.close();  } } |

### 2.5mybatis注解开发和编写dao实现类

1. 删除resources下的dao的mapper文件
2. 在IUserDao接口方法上添加注解
3. 修改SqlMapConfig，映射到IUserDao接口类

|  |
| --- |
| **public interface** IUserDao {  */\*\*  \* 查询所有操作  \** ***@return*** *\*/* @Select(**"select \* from user"**)  List<User> findAll(); } |
| *<!--指定映射配置文件的位置，映射配置文件指的是每个dao独立的配置文件  如果是用注解来配置的话，此处应该使用class属性指定被注解的dao全类名 -->* <**mappers**>  *<!--<mapper resource="com/byf/dao/IUserDao.xml"></mapper>-->* <**mapper class="com.byf.dao.IUserDao"**></**mapper**> </**mappers**> |

Mybatis入门案例：

1. 读取配置文件；
2. 创建SqlSessionFactory工厂；
3. 创建SqlSession对象；
4. 创建Dao接口的代理对象实现类；
5. 执行Dao的接口方法；
6. 关闭/释放SqlSession资源。

|  |
| --- |
| **public class** UserDaoImpl **implements** IUserDao {  **private** SqlSessionFactory **factory**;   **public** UserDaoImpl(SqlSessionFactory factory){  **this**.**factory** = factory;  }  **public** List<User> findAll() {  *// 1.使用工程创建SqlSession对象* SqlSession session = **factory**.openSession();  *// 2.使用session执行查询* List<User> users = session.selectList(**"com.byf.dao.IUserDao.findAll"**);  *// 3.返回查询结果* session.close();  **return** users;  } } |

|  |
| --- |
| **public class** MybatisTest {  @Test  **public void** testConfig() **throws** IOException {  */\*\*  \* 1.读取配置文件  \* 1）绝对路径：d:/xxx/xxx.xml 不可用  \* 2）相对路径：src/java/main/xxx.xml 不可用  \* 3）使用类加载器，他只能兑取类路径的配置文件  \* 4）使用ServletContext对象的getRealPath()  \*/* InputStream in = Resources.*getResourceAsStream*(**"SqlMapConfig.xml"**);   */\*\*  \* 2.创建SqlSessionFactory工厂  \* 1）创建工厂mybatis使用了构建者模式  \* 优势：把对象创建的细节。因此，使用者直接调用对象  \*/* SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  SqlSessionFactory factory = builder.build(in);   */\*\*  \* 4.使用SqlSession创建Dao接口的代理对象  \* 1）创建Dao接口实现类使用了代理模式  \* 优势：不修改源码的基础上，增强方法  \*/* IUserDao userDao = **new** UserDaoImpl(factory);   *//5.使用代理对象执行方法* List<User> users = userDao.findAll();  **for** (User user:users){  System.***out***.println(user);  }  *//6.释放资源* in.close();  } } |
| **19:37,079 [ main] DEBUG source.pooled.PooledDataSource - PooledDataSource forcefully closed/removed all connections.**  **19:37,349 [ main] DEBUG ansaction.jdbc.JdbcTransaction - Opening JDBC Connection**  **19:37,749 [ main] DEBUG source.pooled.PooledDataSource - Created connection 249155636.**  **19:37,749 [ main] DEBUG ansaction.jdbc.JdbcTransaction - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@ed9d034]**  **19:37,759 [ main] DEBUG com.byf.dao.IUserDao.findAll - ==> Preparing: select \* from user**  **19:37,799 [ main] DEBUG com.byf.dao.IUserDao.findAll - ==> Parameters:**  **19:37,889 [ main] DEBUG com.byf.dao.IUserDao.findAll - <== Total: 6**  **19:37,889 [ main] DEBUG ansaction.jdbc.JdbcTransaction - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection@ed9d034]**  **19:37,889 [ main] DEBUG ansaction.jdbc.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@ed9d034]**  **19:37,889 [ main] DEBUG source.pooled.PooledDataSource - Returned connection 249155636 to pool.**  **User(id=41, username=Jack, birthday=Tue Feb 27 17:47:08 CST 2018, sex=男, address=北京)** |