Mybatis框架课程第一天

第1章 框架概述

1.1 什么是框架

1.1.1 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。

框架使用一般都应该降版本使用

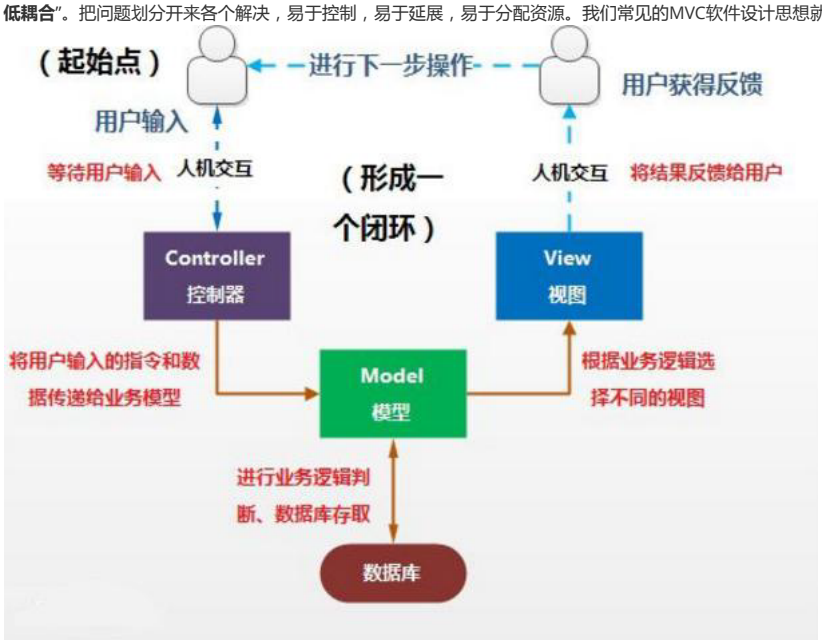
1.1.2 框架要解决的问题

框架要解决的最重要的一个问题是技术整合的问题，在J2EE的 框架中，有着各种各样的技术，不同的软件企业需要从J2EE中选择不同的技术，这就使得软件企业最终的应用依赖于这些技术，技术自身的复杂性和技术的风险性将会直接对应用造成冲击。而应用是软件企业的核心，是竞争力的关键所在，因此应该将应用自身的设计和具体的实现技术**解耦。**这样，软件企业的研发将集中在应用的设计上，而不是具体的技术实现，技术实现是应用的底层支撑，它不应该直接对应用产生影响。

框架一般处在低层应用平台（如J2EE）和高层业务逻辑之间的中间层。

1.1.3 软件开发的分层重要性

框架的重要性在于它实现了部分功能，并且能够很好的将低层应用平台和高层业务逻辑进行了缓和。为了实现软件工程中的"**高内聚、低耦合**"。把问题划分开来各个解决，易于控制，易于延展，易于分配资源。我们常见的MVC软件设计思想就是很好的分层思想。



通过分层更好的实现了各个部分的职责，在每一层将再细化出不同的框架，分别解决各层关注的问题。

1.1.4 分层开发下的常见框架

常见的JavaEE开发框架：

1、解决数据的持久化问题的框架

MyBatis 编辑

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github。

iBATIS一词来源于"internet"和"abatis"的组合，是一个基于Java的持久层框架。iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAOs）

作为持久层的框架，还有一个封装程度更高的框架就是Hibernate，但这个框架因为各种原因目前在国内的流行程度下降太多，现在公司开发也越来越少使用。目前使用Spring Data来实现数据持久化也是一种趋势。

2、解决WEB层问题的MVC框架

spring MVC 编辑

Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。Spring 框架提供了构建 Web 应用程序的全功能 MVC 模块。使用 Spring 可插入的 MVC 架构，从而在使用Spring进行WEB开发时，可以选择使用Spring的SpringMVC框架或集成其他MVC开发框架，如Struts1(现在一般不用)，Struts2等。

3、解决技术整合问题的框架

spring框架 编辑

Spring框架是由于软件开发的复杂性而创建的。Spring使用的是基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅仅限于服务器端的开发。从简单性、可测试性和松耦合性角度而言，绝大部分Java应用都可以从Spring中受益。

◆目的：解决企业应用开发的复杂性

◆功能：使用基本的JavaBean代替EJB，并提供了更多的企业应用功能

◆范围：任何Java应用

Spring是一个轻量级控制反转(IoC)和面向切面(AOP)的容器框架。

1.1.5 MyBatis框架概述

mybatis是一个优秀的基于java的持久层框架，它内部封装了jdbc，使开发者只需要**关注sql语句**本身，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。

mybatis通过**xml**或注解的方式将要执行的各种**statement**配置起来，并通过java对象和statement中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回。

采用ORM思想解决了实体和数据库映射的问题，对jdbc进行了封装，屏蔽了jdbc api底层访问细节，使我们不用与jdbc api打交道，就可以完成对数据库的持久化操作。

1.2 JDBC编程的分析

1.2.1 jdbc程序的回顾

```
public static void main(String[] args) {
    Connection connection = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
    try {
        //加载数据库驱动
        Class.forName("com.mysql.jdbc.Driver");
        //通过驱动管理类获取数据库链接
        connection = DriverManager .getConnection("jdbc:mysql://localhost:3306/mybatis?
characterEncoding=utf-8","root", "root");
        //定义sql语句 ?表示占位符
        String sql = "select * from user where username = ?";
        //获取预处理
        statement preparedStatement = connection.prepareStatement(sql);
        //设置参数，第一个参数为sql语句中参数的序号（从1开始），第二个参数为设置的参数值
preparedStatement.setString(1, "王五");
        //向数据库发出sql执行查询，查询出结果集
        resultSet = preparedStatement.executeQuery();
        //遍历查询结果集
        while(resultSet.next()){
            System.out.println(resultSet.getString("id")+" "+resultSet.getString("username"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }finally{
        //释放资源
        if(resultSet!=null){
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(preparedStatement!=null){
            try {
                preparedStatement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(connection!=null){
            try {
                connection.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
//上边使用jdbc的原始方法（未经封装）实现了查询数据库表记录的操作。
```

1.2.2 jdbc问题分析

1、数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

2、Sql语句在代码中硬编码，造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

3、使用preparedStatement向占有位符号传参数存在硬编码，因为sql语句的where条件不一定，可能多也可能少，修改sql还要修改代码，系统不易维护。

4、对结果集解析存在硬编码（查询列名），sql变化导致解析代码变化，系统不易维护，如果能将数据库记录封装成pojo对象解析比较方便。

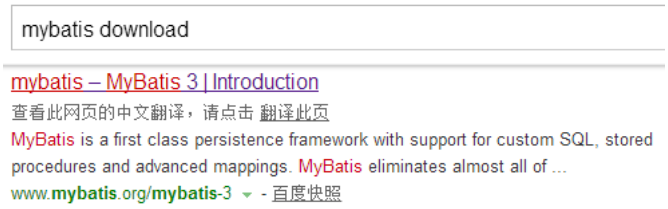
第2章 Mybatis框架快速入门

通过前面的学习，我们已经能够使用所学的基础知识构建自定义的Mybatis框架了。这个过程是基本功的考验，我们已经强大了不少，但现实是残酷的，我们所定义的Mybatis框架和真正的Mybatis框架相比，还是显得渺小。行业内所流行的Mybatis框架现在我们将开启学习。

2.1 Mybatis框架开发的准备

2.1.1 官网下载Mybatis框架

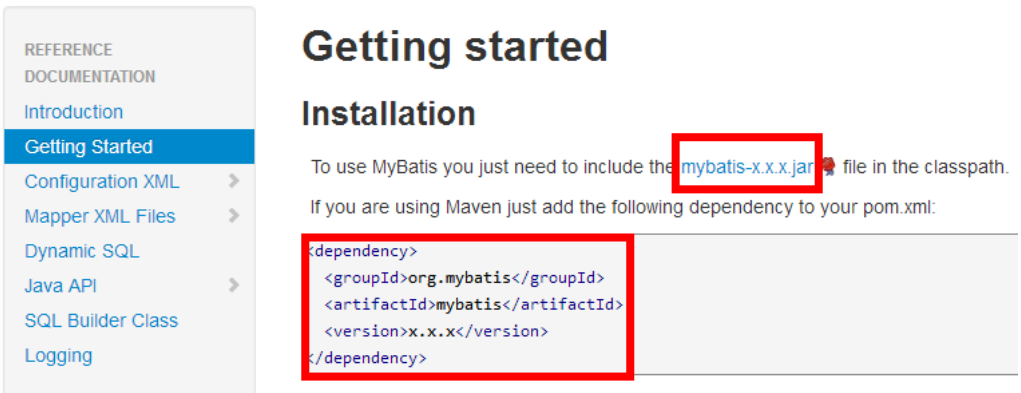
从百度中“mybatis download”可以下载最新的Mybatis开发包。



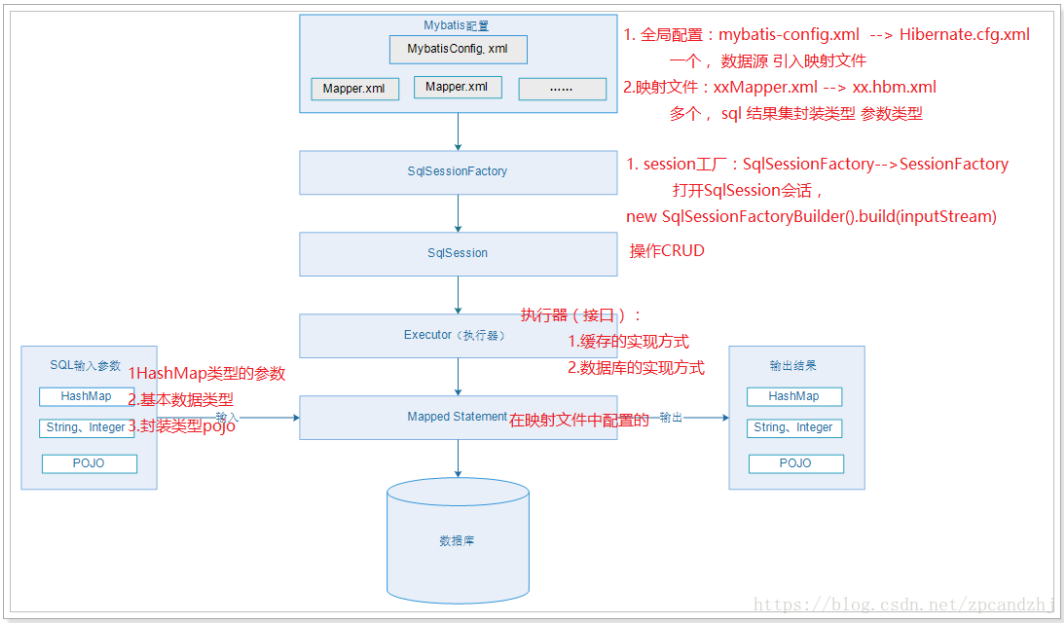
进入选择语言的界面，进入中文版本的开发文档。

我们可以看到熟悉的中文开发文档了。

下载相关的jar包或maven开发的坐标。



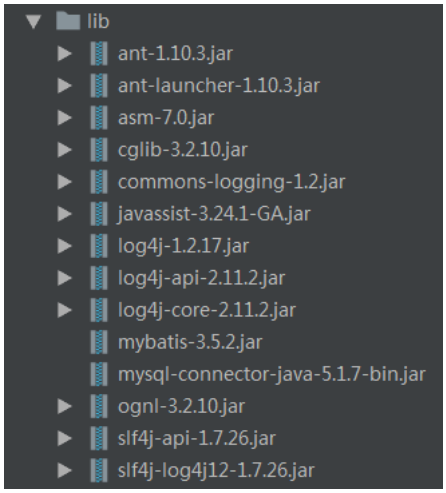
Mybaits整体架构



2.2 搭建Mybatis开发环境

2.2.1 创建Java工程

2.2.2 添加Mybatis的jar，并准备数据库



在数据库中运行user.sql文件

2.2.3 编写User实体类：和数据库的字段一一对应

```
package com.yaorange.entity;

import java.util.Date;

/**
 * 用户的实体类
 */
public class User {
    private int id;
    private String username;// 用户姓名
    private String sex;// 性别
    private Date birthday;// 生日
    private String address;// 地址

    public int getId() {
        return id;
    }

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

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
```

```
        this.username = username;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", sex='" + sex + '\'' +
                ", birthday=" + birthday +
                ", address='" + address + '\'' +
                '}';
    }
}
```

2.2.4 编写持久层接口UserDao

```
package com.yaorange.mapper;

import com.yaorange.entity.User;
import java.util.List;

/**
 * UserDao接口就是我们的持久层接口（也可以写成UserMapper）
 */
public interface UserDao {
    /**
     * 查询所有用户
     * @return
     */
    List<User> findAll();
}
```

2.2.5 编写持久层接口的映射文件UserDao.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:根标签，namespace：命名空间，随便写，一般保证命名空间唯一 -->
<mapper namespace="test">
    <!-- statement，内容：
      sql语句：
        id：唯一标识，随便写，在同一个命名空间下保持唯一
        resultType：sql语句查询结果集的封装类型,tb_user即为数据库中的表
    -->
    <select id="findAll" resultType="com.yaorange.entity.User">
        select * from user
    </select>
</mapper>
```

2.2.6 编写SqlMapConfig.xml配置文件

```
<?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>
    <!--加载外部属性文件-->
    <properties resource="jdbcConfig.properties"/>
```

```
    <!--定义别名-->
    <typeAliases>
        <typeAlias type="com.yaorange.entity.User" alias="User"/>
    </typeAliases>
    <!-- 配置mybatis的环境 -->
    <environments default="mysql">
        <!-- 配置mysql的环境 -->
        <environment id="mysql">
            <!-- 配置事务的类型 -->
            <transactionManager type="JDBC"/>
            <!-- 配置连接数据库的信息：用的是数据源(连接池) -->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="${jdbc.username}"/>
                <property name="password" value="${jdbc.password}"/>
            </dataSource>
        </environment>
    </environments>
    <!-- 加载mybatis映射配置 -->
    <mappers>
        <mapper resource="sqlmap/UserDao.xml"/>
    </mappers>
</configuration>
```

2.2.7 编写测试类

```
import com.yaorange.entity.User;
import com.yaorange.mapper.UserDao;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;

public class Main {

    public static void main(String[] args) throws IOException {
        //1.读取配置文件
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");
        //2.创建SqlSessionFactory的构建者对象
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        //3.使用构建者创建工厂对象SqlSessionFactory
        SqlSessionFactory factory = builder.build(in);
        // 4.使用SqlSessionFactory生产SqlSession对象
        SqlSession session = factory.openSession();
        // 5.使用SqlSession操作数据库
        List<User> users = session.selectList("test.findAll");
        for(User user : users) {
            System.out.println(user);
        }
        //6.释放资源
        session.close();
        in.close();
    }
}
```

总结：

开发步骤：

1.添加jar：mybatis本身的jar和框架依赖的jar

2.创建数据库，并在java中定义与其映射关联的实体类（属性和数据库字段一一对应，名称和数据类型应该对应），实体中的数据类型只能使用包装类型和自定义类型（数据库中每个字段存在空的概念，如果使用基本数据类型，那么成员变量存在默认值，无法描述空状态）

3.创建mybatis的全局配置文件，在配置文件中配置环境，环境中指定事务处理方式和数据源，同时可以在配置中配置外部属性文件的加载和类别名的指定，必须指定映射文件的加载

4.定义dao的接口类，定义各种操作数据库的接口方法

5.创建dao接口映射文件，在映射文件中定义各种操作数据库的语句，使用标记

```
/***
* @description: 获取当个用户
* @param id
* @return com.yaorange.entity.User
* @exception
*/
```

```
User findOneById(Integer id);
```

在映射文件中定义语句：

```
<select id="findOneById" parameterType="java.lang.Integer" resultType="User">
    select * FROM user WHERE id=#{id}
</select>
```

测试方法：

```
@Test
public void findUserById() throws IOException {
    //1.加载全局配置文件
```

```
        InputStream resource = Resources.getResourceAsStream("SqlMapConfig.xml");
        //2.创建SqlSessionFactory的构建者对象
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        //3.通过构造者对象构造工厂
        SqlSessionFactory factory = builder.build(resource);
        //4.通过工厂获取Sql
        // Session对象
        SqlSession session = factory.openSession();
        //5.通过session操作数据库
        //方法的第一个参数是在映射文件中的namespace.语句的id
        List<User> users = session.selectList("test.findOneById",44);
        for (User user : users){
            System.out.println(user);
        }
        //6.释放资源
        session.close();
        resource.close();
    }
```

2.4新增操作

定义接口方法

```
    /***
    * @description: 新增用户
    * @param user
    * @return void
    * @exception
    */
    void add(User user);
```

定义映射语句：

```
<insert id="add" parameterType="User">
        insert into user(username,birthday,sex,address) value(#{username},#{birthday},#{sex},#{address})
    </insert>
```

编写测试：所有**更新操作都需要进行事务提交**

```
    @Test
    public void add() throws IOException {
        //1.加载全局配置文件
        InputStream resource = Resources.getResourceAsStream("SqlMapConfig.xml");
        //2.创建SqlSessionFactory的构建者对象
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        //3.通过构造者对象构造工厂
        SqlSessionFactory factory = builder.build(resource);
        //4.通过工厂获取Sql
        // Session对象
        SqlSession session = factory.openSession();
        //5.通过session操作数据库
        //方法的第一个参数是在映射文件中的namespace.语句的id
        User user = new User();
        user.setUsername("小王");
        user.setBirthday(new Date());
        user.setSex("女");
        user.setAddress("四川成都");

        int num = session.insert("test.add", user);
        System.out.println(num);

        //提交事务
        session.commit();
        //6.释放资源
        session.close();
        resource.close();
    }
```

2.5更新操作：

映射语句：

```
<update id="updateUser" parameterType="User">
        update user set username=#{username},birthday=#{birthday},sex=#{sex},address=#{address} where id=#{id}
    </update>
```

测试代码：

```
    @Test
    public void update() throws IOException {
        //1.加载全局配置文件
        InputStream resource = Resources.getResourceAsStream("SqlMapConfig.xml");
        //2.创建SqlSessionFactory的构建者对象
```

```
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        //3.通过构造者对象构造工厂
        SqlSessionFactory factory = builder.build(resource);
        //4.通过工厂获取Sql
        // Session对象
        SqlSession session = factory.openSession();
        //5.通过session操作数据库
        //方法的第一个参数是在映射文件中的namespace.语句的id
        User user = new User();

        user.setId(46);

        user.setUsername("小王");
        user.setBirthday(new Date());
        user.setSex("男");
        user.setAddress("四川雅安");

        int num = session.update("test.updateUser", user);
        System.out.println(num);

        //提交事务
        session.commit();
        //6.释放资源
        session.close();
        resource.close();
    }
```

2.6删除操作

映射语句：

```
<delete id="deleteUserById" parameterType="int">    delete from user where id=#{id}</delete>
```

测试代码：

```
    @Test
    public void delete() throws IOException {
        //1.加载全局配置文件
        InputStream resource = Resources.getResourceAsStream("SqlMapConfig.xml");
        //2.创建SqlSessionFactory的构建者对象
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        //3.通过构造者对象构造工厂
        SqlSessionFactory factory = builder.build(resource);
        //4.通过工厂获取Sql
        // Session对象
        SqlSession session = factory.openSession();
        //5.通过session操作数据库
        //方法的第一个参数是在映射文件中的namespace.语句的id
        int num = session.delete("test.deleteUserById", 46);
        System.out.println(num);

        //提交事务
        session.commit();
        //6.释放资源
        session.close();
        resource.close();
    }
```

补充：

添加log4j的配置：

将log4j属性文件添加到classpath（项目的根目录）目录下。

#{}和${}的区别：

${}是字符串拼接

#{}是占位符

建议：一般不建议使用 ${},会引起注入攻击

selectOne和SelectList：

如果查询结果是一条数据，那么两种都可以使用

如果查询结果是多条数据，那么只能使用selectList，使用SelectOne会出现异常报错

parameterType：

指定输入的参数数据类型

resultType:

指定输出结果的映射类型