0. 前提： idea关联maven

Maven 是一个项目管理工具，可以对 Java 项目进行自动化的构建和依赖管理

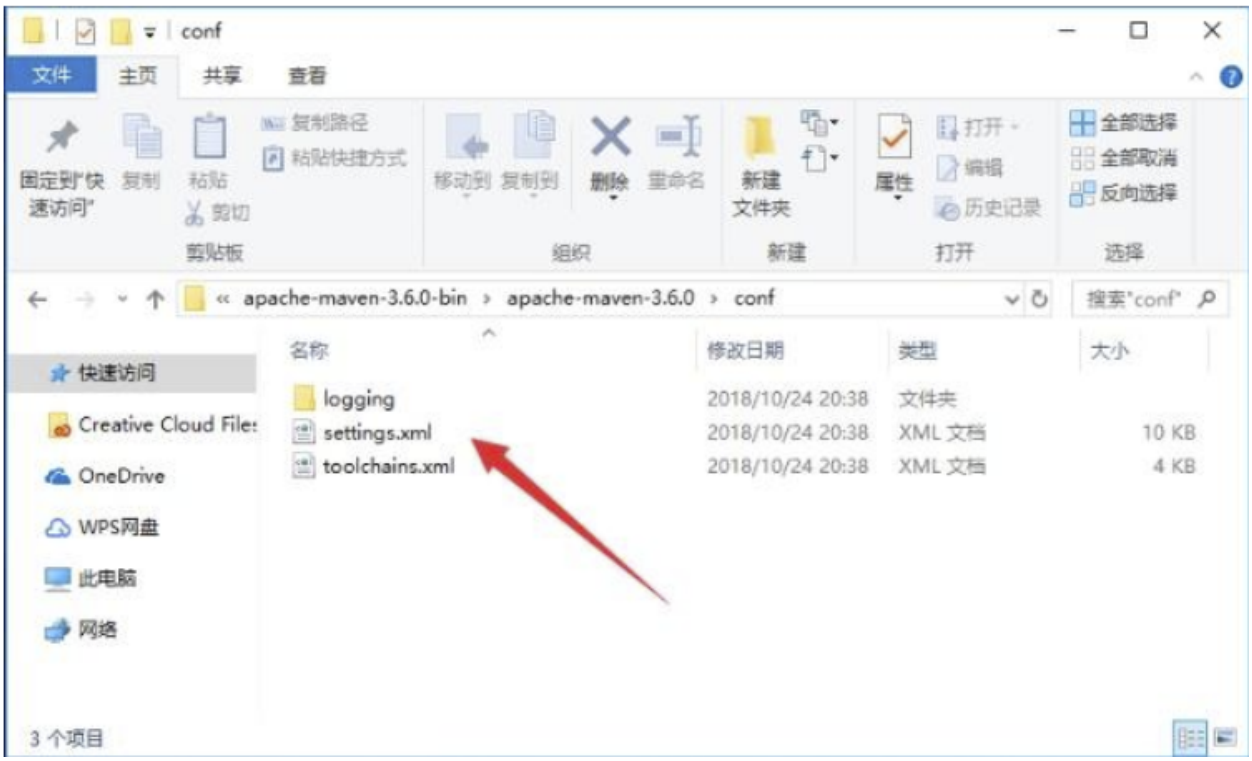
1. Maven下载地址：http://maven.apache.org/download.cgi

	Link
Binary tar.gz archive	apache-maven-3.6.0-bin.tar.gz
Binary zip archive	apache-maven-3.6.0-bin.zip
Source tar.gz archive	apache-maven-3.6.0-src.tar.gz
Source zip archive	apache-maven-3.6.0-src.zip

下载好之后，解压并选择存放路径，**注意不要放在中文路径底下**；也可以使用我给你们提供的压缩包，解压并选择存放路径

2. 配置Maven本地仓库路径

   进入Maven存放路径，进入conf文件夹，打开setting.xml文件



   找到，把注释去掉或重新加入一行（自己的Maven仓库路径）

   D:\soft\repository

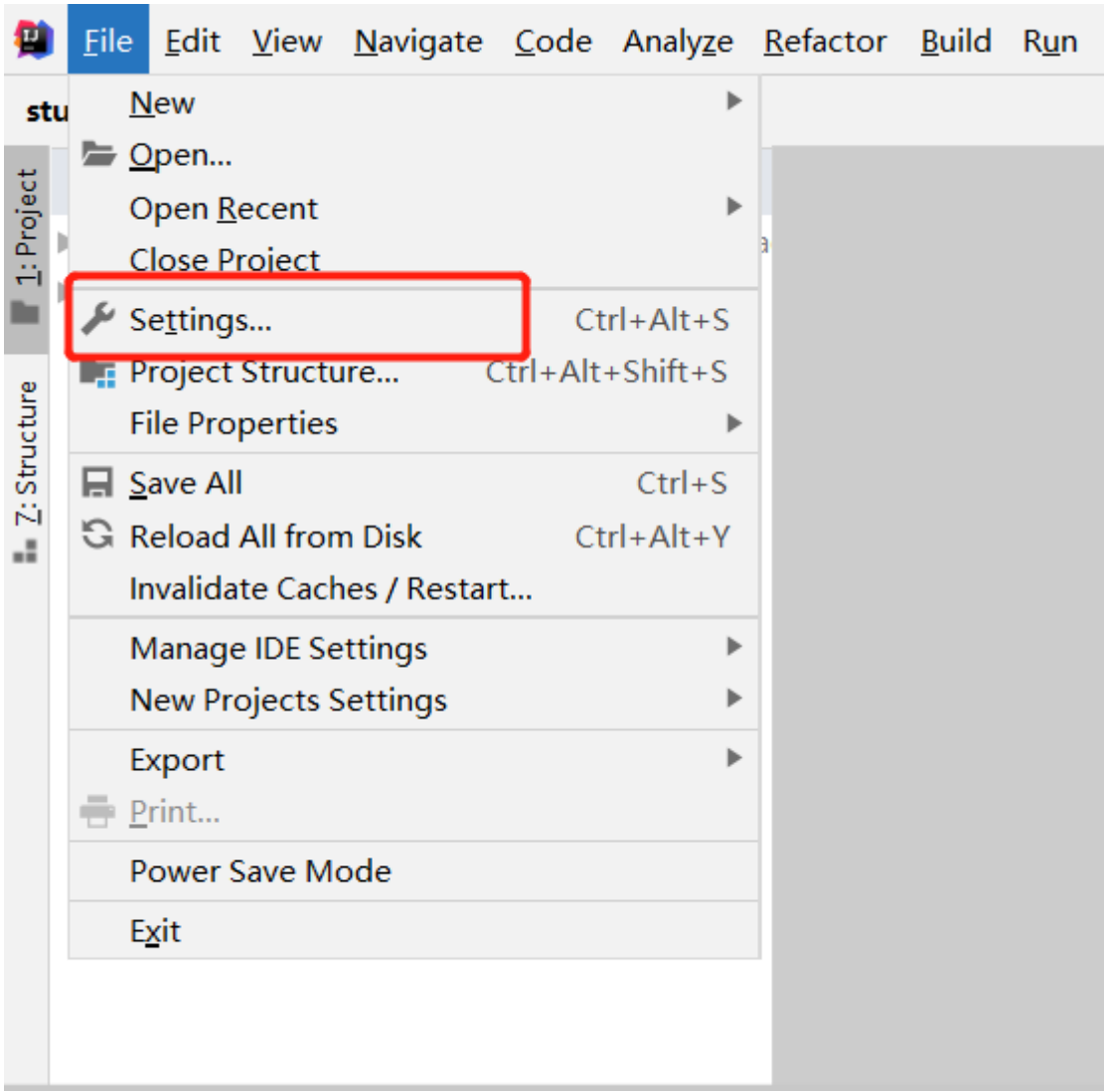
```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
   | The path to the local repository maven will use to store artifacts.
   |
   | Default: ${user.home}/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  -->
  <localRepository>D:\soft\repository</localRepository>
```

3. 配置Maven镜像源

   找到，里面配置上maven的阿里云镜像源

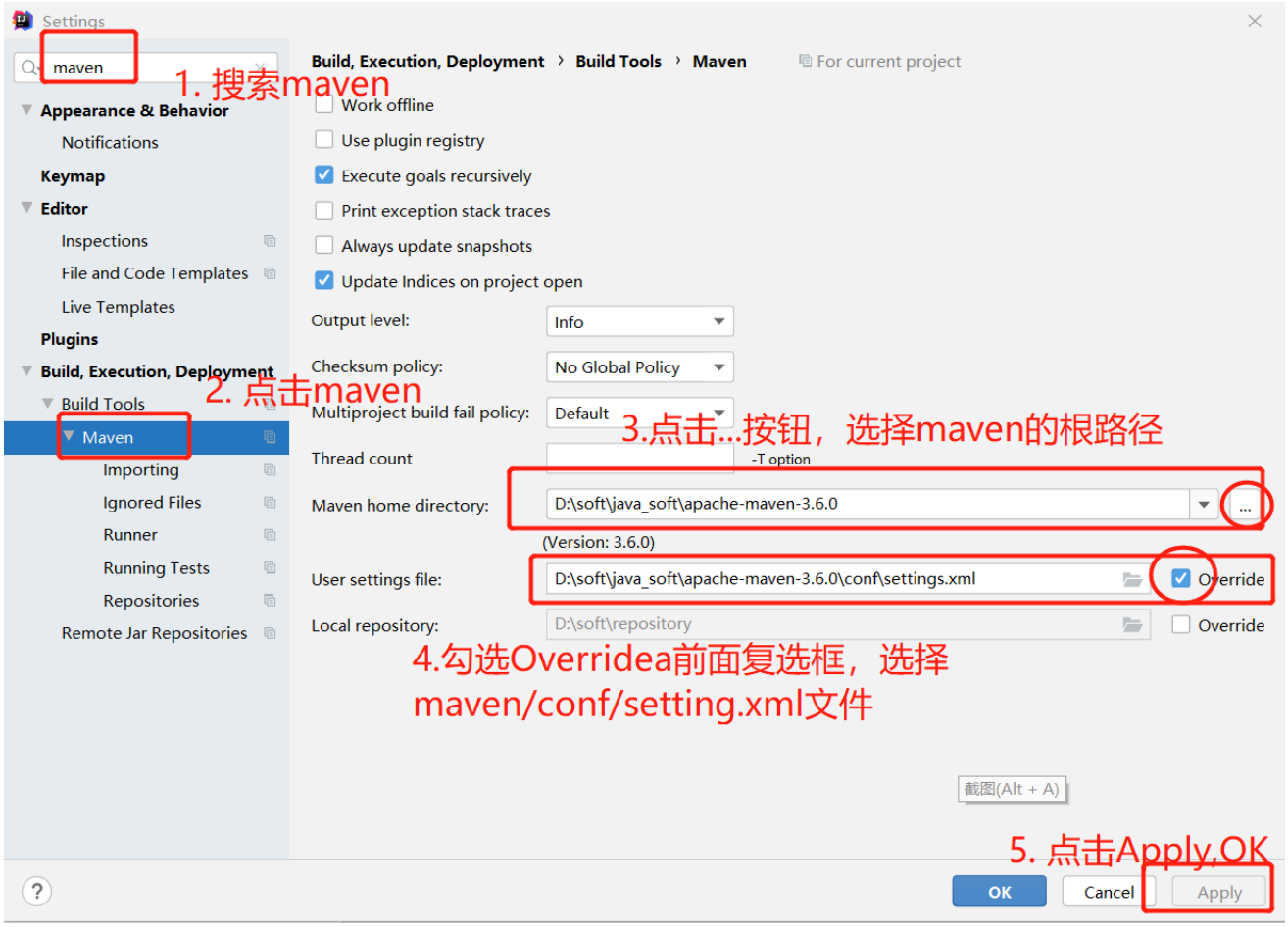
```
<mirrors>
    <!-- 阿里云镜像 -->
    <mirror>
        <id>alimaven</id>
        <name>aliyun maven</name>
        <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
        <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>
```

4. 接下来打开IDEA，进入下方界面，选择右下角File->Settings



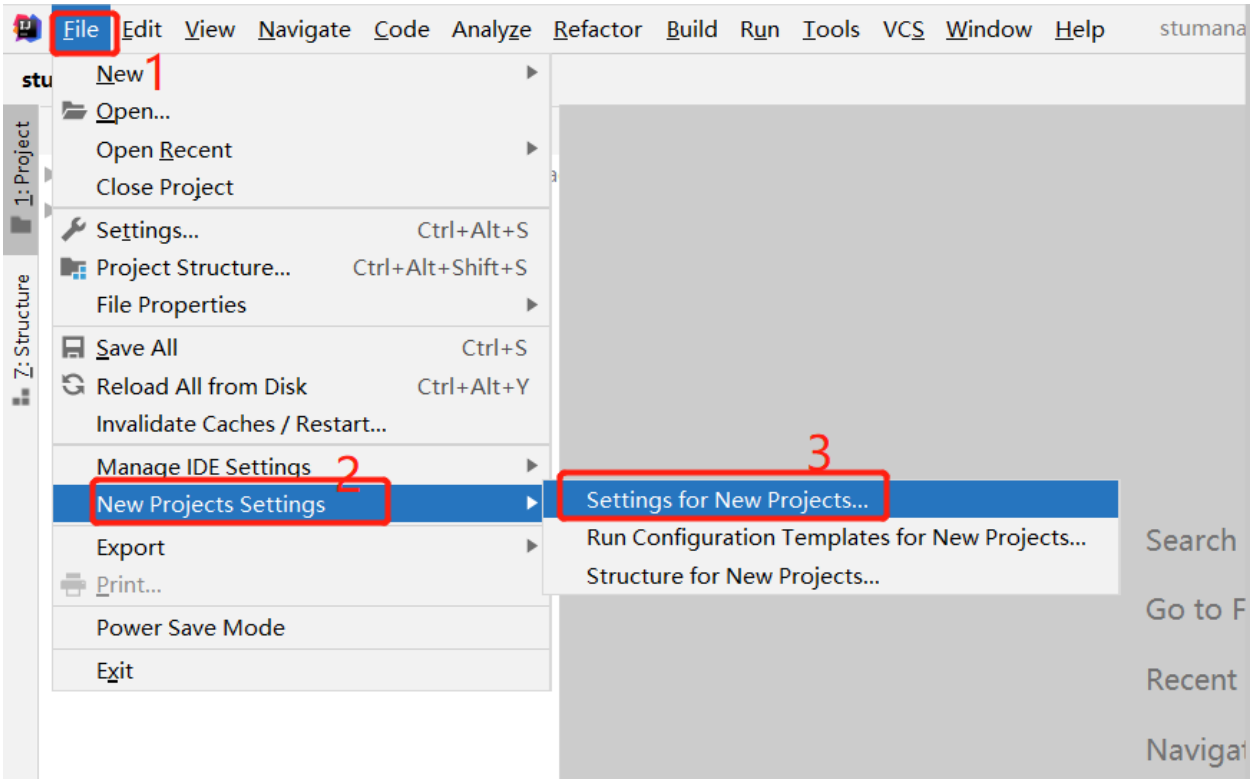
复选框输入maven搜索-----> 点击左边的maven菜单------->右边部分 选择 Maven home directory 到自己的maven的根目录

---------> 配置User settings file: 勾选Override前面复选框，选择maven根路径/conf/setting.xml文件；下面的Local repository会自动更新 ---------->最后点击apply按钮，点击OK按钮

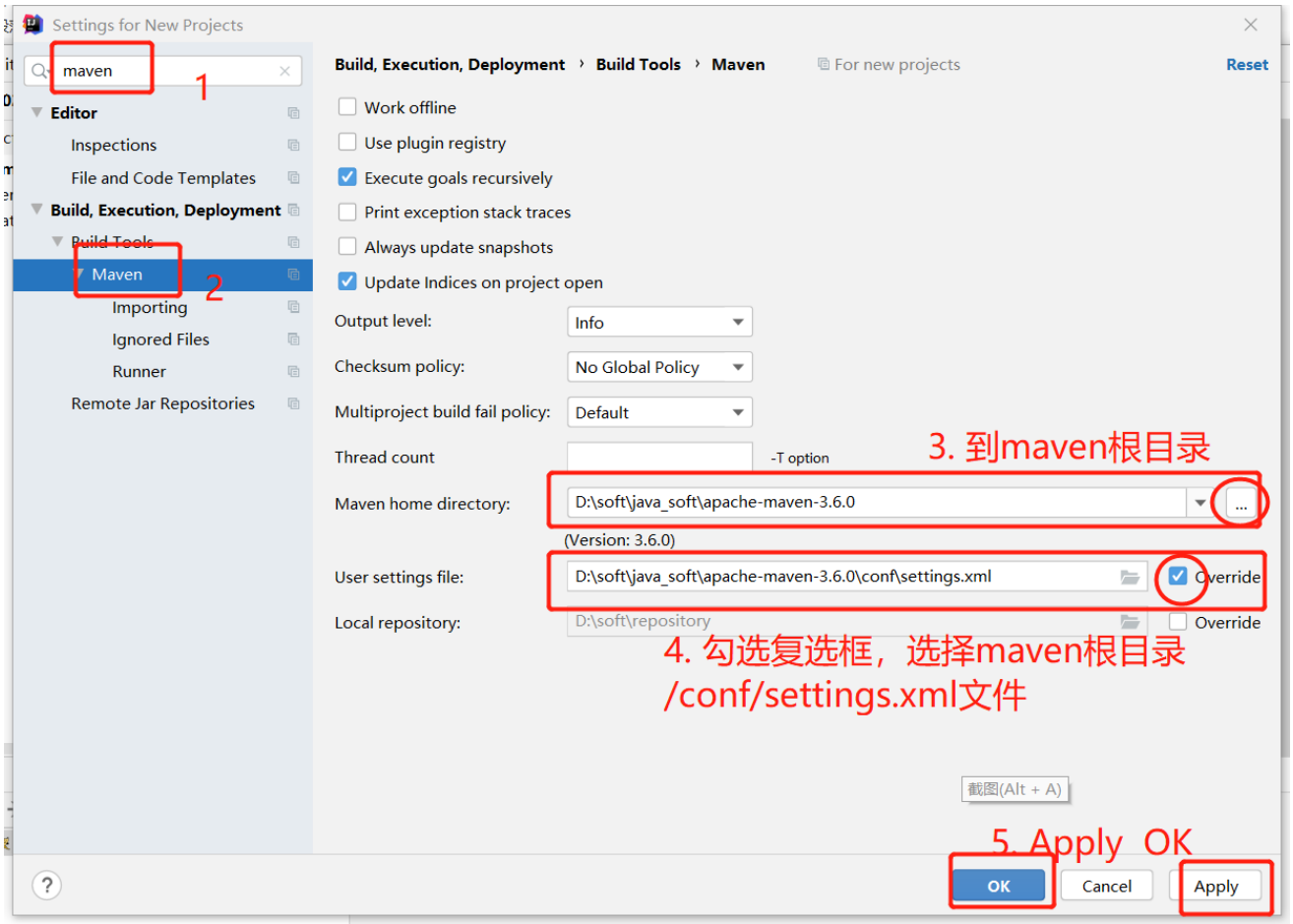


5. 上述的配置仅仅当前项目生效，要想以后的项目全部生效，需要全局配置

   选择 File 下的 New Projects Settings 下的Settings for New Projects...



进入后跟上一步步骤配置方式一样



此时，maven配置成功

spring笔记

1. spring环境搭建

核心概念：

IOC: 控制翻转 DI: 依赖注入

IOC的目的就是依赖注入；DI的前提就是控制反转

这两个概念在实现环境搭建效果之后再另行讲解

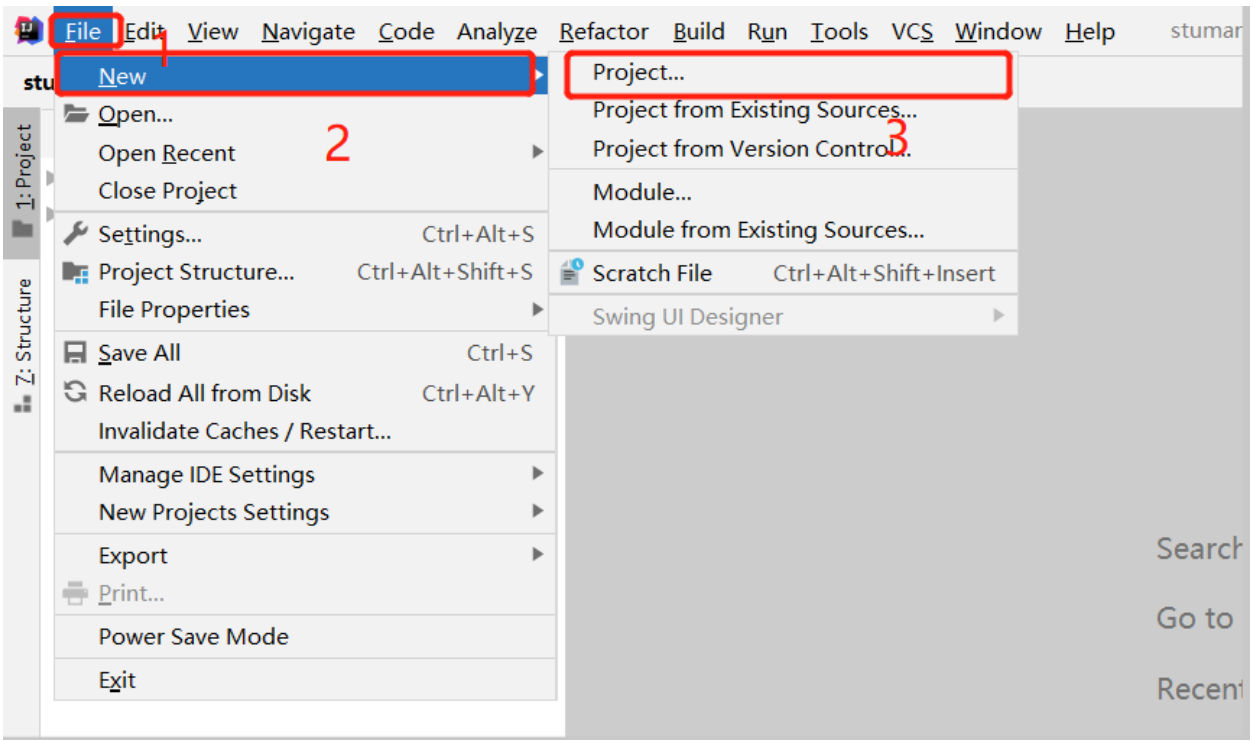
1. 创建maven工程

   注意： **使用maven工程的时候电脑必要要连接外网**

   maven工程有maven java project 和maven web project之分； 在这里面咱们直接创建maven web project:

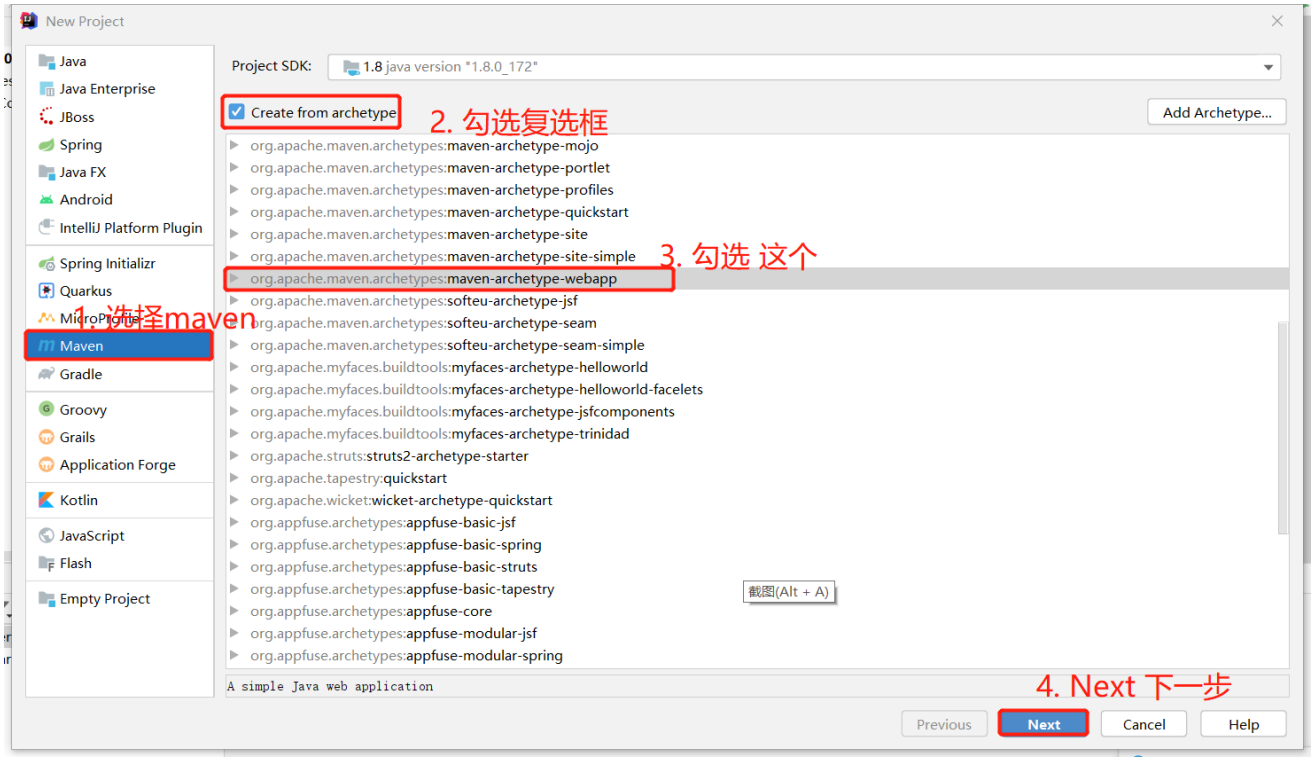
   创建maven web project项目流程：

   选择File----->New ------>Project...



选择左边的Maven--------->右边 Create from archetype前面的复选框勾选上------->选中下面的maven-archetype-webapp骨架

---------->Next （下一步）

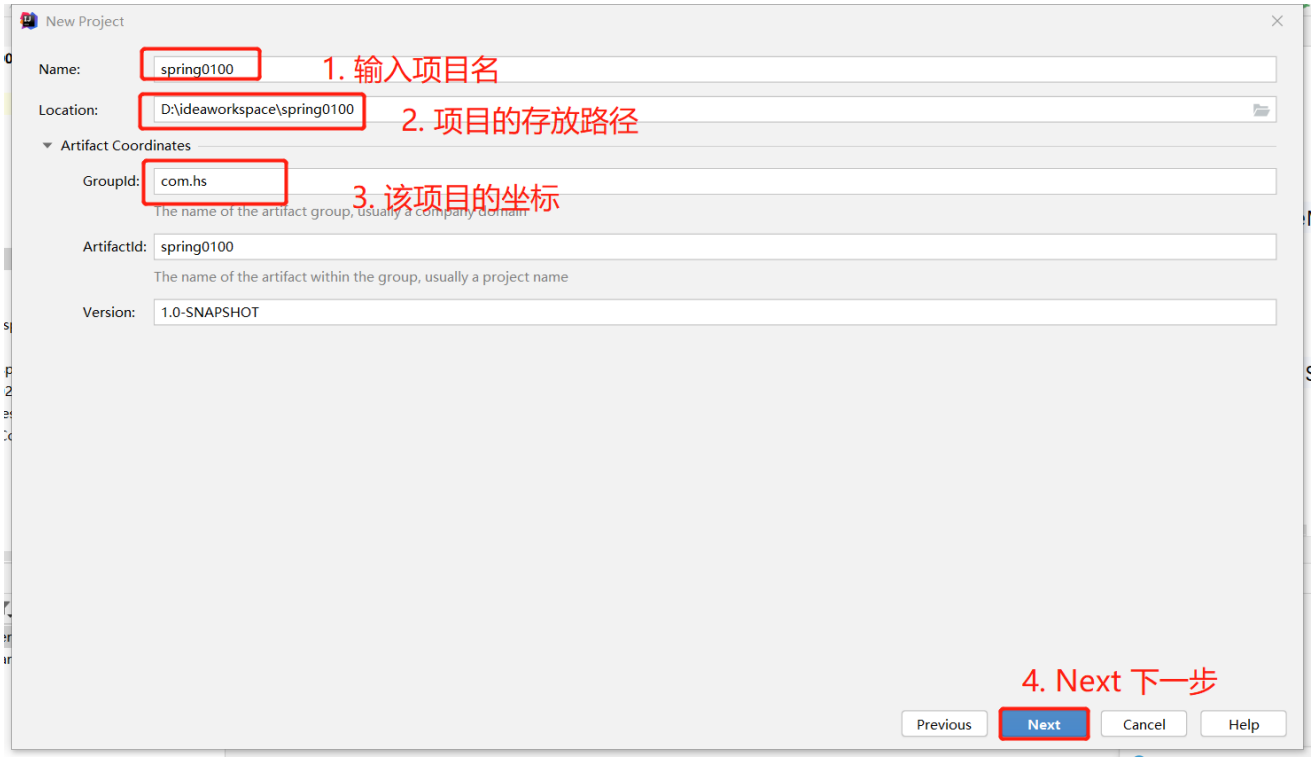


输入项目名，选择项目的存放路径，可以输入 当前项目的GroupId，最后Next下一步

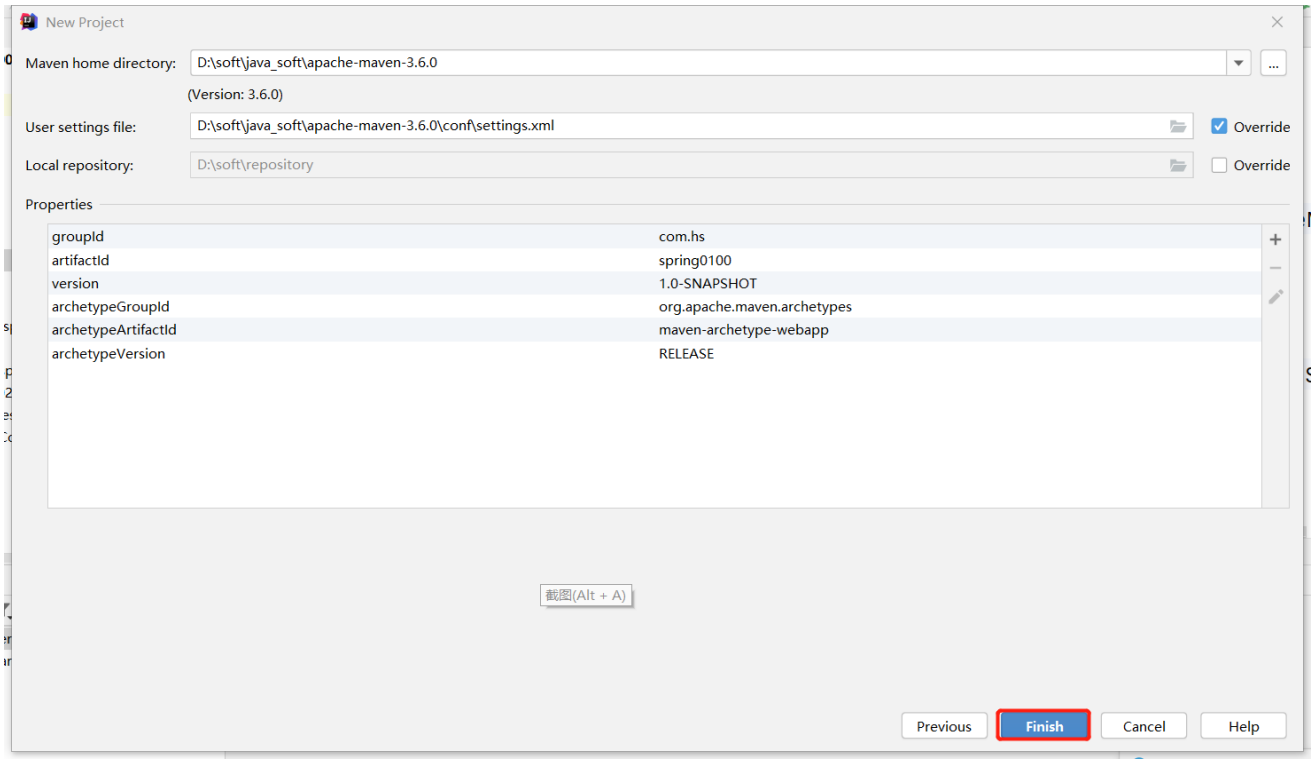
groupid和artifactId被统称为“坐标”是为了保证项目唯一性而提出的，如果你要把你项目弄到maven本地仓库去，你想要找到你的 项目就必须根据这两个id去查找。

groupId一般分为多个段，这里只说两段，第一段为域，第二段为公司名称。域又分为org、com、cn等等许多，其中org为非营利 组织，com为商业组织。

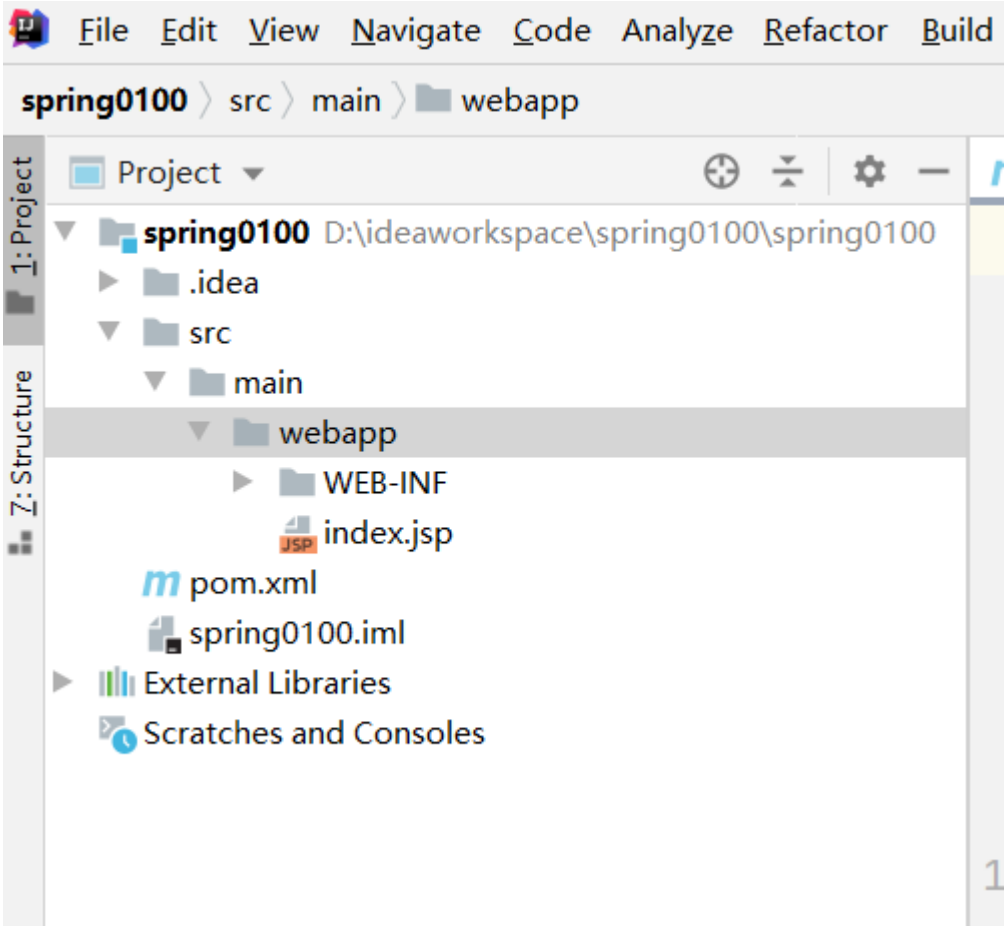
ArtifactID就是项目的唯一的标识符，实际对应项目的名称，就是项目根目录的名称。比如我创建一个项目，我一般会将groupId设 置为com.ys，com表示域，ys是我个人姓名缩写，**Artifact Id**设置为hellomaven，表示你这个项目的名称是hellomaven



如果之前maven关联好的话，直接点Finish,项目就创建成功

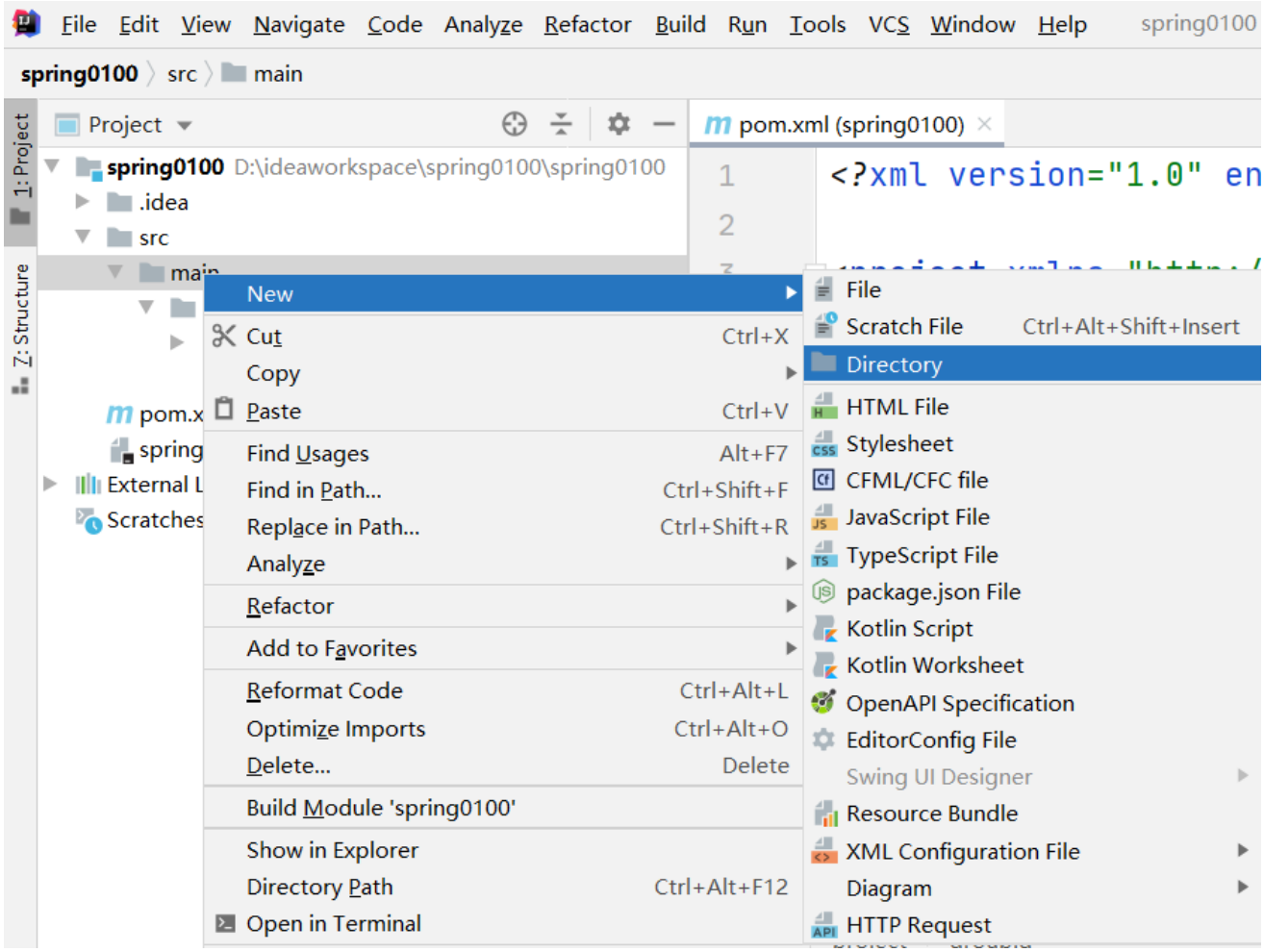


创建好的项目的目录结构：

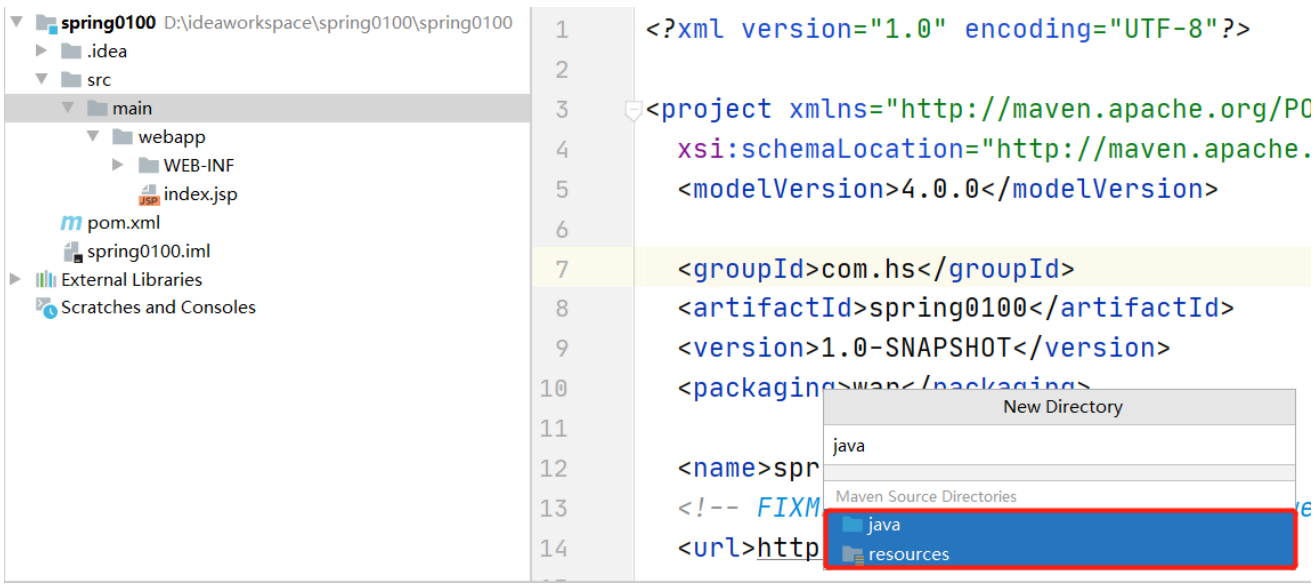


此时这个项目的目录结构缺少几个目录，咱们手动创建把它补全；

在main底下创建 java和resource目录； 选中main,右键 New----Directory..



java和resource目录都选中，回车创建即可



同样的方法在src目录底下创建出 src/test/java; src/test/resources 目录；在此就不再上图演示了

还有 新创建好的pom.xml文件内容是这样的：

咱们需要吧 从这开始 ----到这结束 中间的内容直接删除掉，我们 不需要； 在下面代码里面已经加上响应的注释

pom.xml文件：

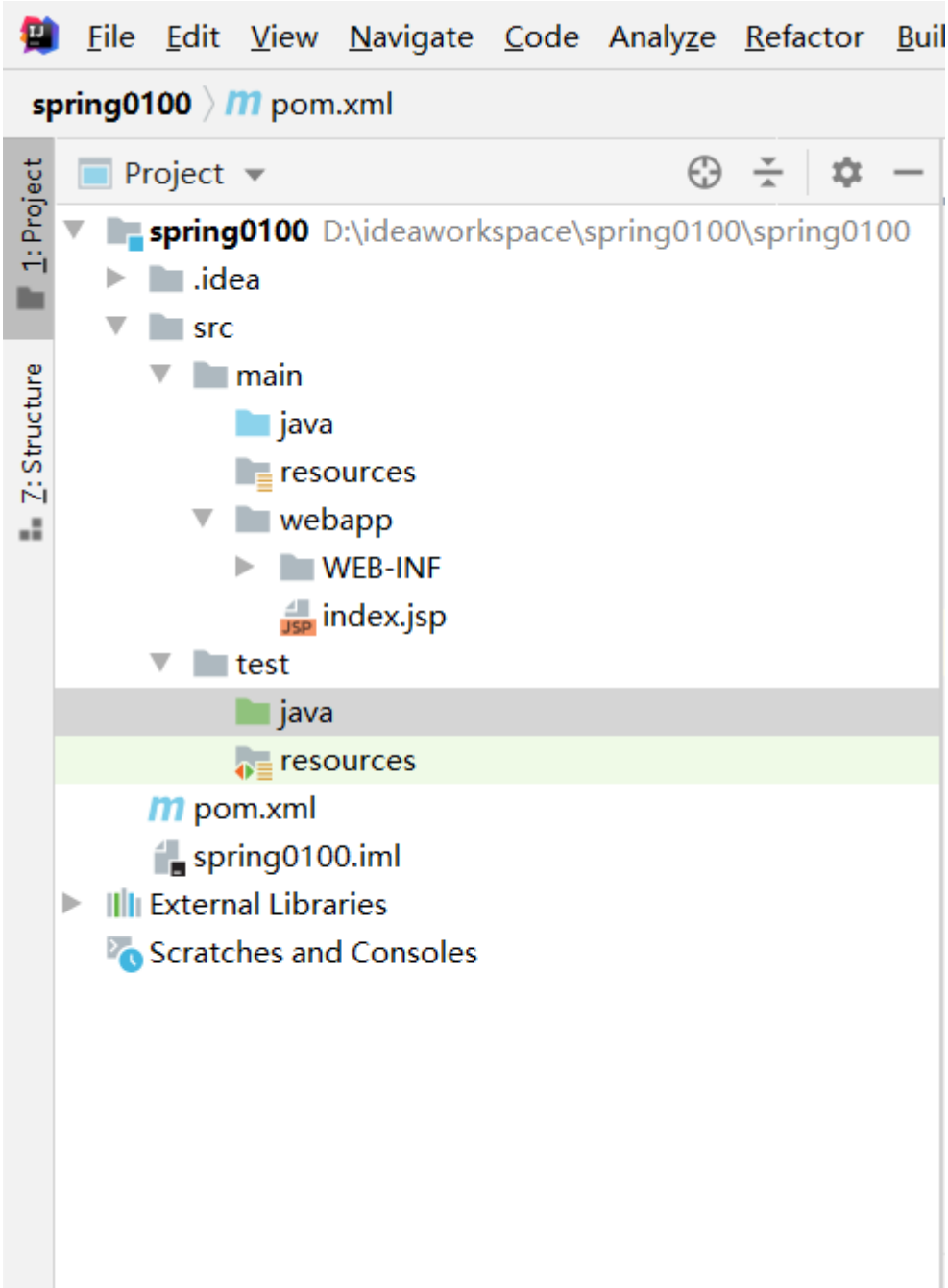
```
   <?xml version="1.0" encoding="UTF-8"?>

   <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>com.hs</groupId>
     <artifactId>spring0100</artifactId>
     <version>1.0-SNAPSHOT</version>
     <packaging>war</packaging>

     <name>spring0100 Maven Webapp</name>
     <!-- FIXME change it to the project's website -->
     <url>http://www.example.com</url>

     <!-- 从这开始 -->
     <properties>
       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
       <maven.compiler.source>1.7</maven.compiler.source>
       <maven.compiler.target>1.7</maven.compiler.target>
     </properties>

     <dependencies>
       <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.11</version>
         <scope>test</scope>
       </dependency>
     </dependencies>

     <build>
       <finalName>spring0100</finalName>
       <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults
(may be moved to parent pom) -->
         <plugins>
           <plugin>
             <artifactId>maven-clean-plugin</artifactId>
             <version>3.1.0</version>
           </plugin>
           <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
           <plugin>
             <artifactId>maven-resources-plugin</artifactId>
             <version>3.0.2</version>
           </plugin>
           <plugin>
             <artifactId>maven-compiler-plugin</artifactId>
             <version>3.8.0</version>
           </plugin>

           <plugin>
```

```
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.1</version>
          </plugin>
          <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.2.2</version>
          </plugin>
          <plugin>
            <artifactId>maven-install-plugin</artifactId>
            <version>2.5.2</version>
          </plugin>
          <plugin>
            <artifactId>maven-deploy-plugin</artifactId>
            <version>2.8.2</version>
          </plugin>
        </plugins>
      </pluginManagement>
    </build>

    <!-- 到这结束 -->
  </project>
```

删掉相应内容的pom.xml文件：

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

  <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.hs</groupId>
    <artifactId>spring0100</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>spring0100 Maven Webapp</name>
    <!-- FIXME change it to the project's website -->
    <url>http://www.example.com</url>

  </project>
```

最终完整的项目目录结构：


File Edit View Navigate Code Analyze Refactor Buil
spring0100 pom.xml
Project
1: Project
7: Structure
spring0100 D:\ideaworkspace\spring0100\spring0100
.idea
src
main
java
resources
webapp
WEB-INF
index.jsp
test
java
resources
pom.xml
spring0100.iml
External Libraries
Scratches and Consoles

目录结构说明：

```
hellomaven 工程
    |---src                    源码
    |---|---main               存放主程序
    |---|---|---java           存放Java源文件
    |---|---|---resources      存放框架或其他工具的配置文件
    |---|---test               存放测试程序
    |---|---|---java           存放Java 测试的源文件
    |---|---|---resources      存放测试的配置文件
    |---pom.xml                Maven 工程的核心配置文件
```

2. pom.xml文件里面导入依赖

```
<properties>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <maven.compiler.source>1.7</maven.compiler.source>
   <maven.compiler.target>1.7</maven.compiler.target>
   <!-- spring版本号 -->
   <spring.version>4.3.14.RELEASE</spring.version>

   <!-- log4j日志文件管理包版本 -->
   <slf4j.version>1.7.22</slf4j.version>
   <log4j.version>1.2.17</log4j.version>
 </properties>

 <dependencies>

   <dependency>
     <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>4.12</version>
     <scope>test</scope>
   </dependency>

   <dependency>
     <groupId>log4j</groupId>
     <artifactId>log4j</artifactId>
     <version>${log4j.version}</version>
   </dependency>
   <!-- spring -->
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-aop</artifactId>
     <version>${spring.version}</version>
   </dependency>

   <dependency>
     <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-aspects</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.13</version>
  </dependency>



  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-expression</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument-tomcat</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>

    <version>${spring.version}</version>
```

```
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jms</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-messaging</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-orm</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-oxm</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-tx</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <!-- springmvc -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
```

```
  </dependencies>
```

3. 编写HelloDao接口

```
package com.hs.dao;

public interface HelloDao {
    public void sayHello();
}
```

4. 编写HelloDaoImpl接口实现类

```
package com.hs.dao.impl;

import com.hs.dao.HelloDao;

public class HelloDaoImpl implements HelloDao {
    @Override
    public void sayHello() {
        System.out.println("hello world");
    }
}
```

5. 创建spring配置文件applicationContext.xml

   注意： 配置文件需要在src/main/resources目录底下

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.3.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

   <bean id="helloDao" class="com.hs.dao.impl.HelloDaoImpl">

    </bean>
</beans>
```

6. 在src/test/java底下创建测试类Test1.java

```
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test1 {
    //正常方式
    @Test
    public void testDao1(){
        HelloDao dao=new HelloDaoImpl();
        dao.sayHello();
    }

    //spring的方式来调用
    @Test
    public void testDao2(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        //参数是bean标签的id值
        HelloDao helloDao = (HelloDao) ctx.getBean("helloDao");
        helloDao.sayHello();
    }

    //spring的方式来调用
    @Test
    public void testDao3(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        //通过Class对象来调用
        HelloDao helloDao = ctx.getBean(HelloDao.class);
        helloDao.sayHello();
    }
}
```

2. 设值注入和构造注入

1. 创建maven工程(同上)
2. pom.xml里面导入依赖（同上)
3. 在src/main/java底下编写实体类User

```
package com.hs.pojo;

public class User {
    private int id;
    private String name;
    private String pass;


    public User(int id, String name, String pass) {
        this.id = id;
```

```
        this.name = name;
        this.pass = pass;
    }

    public User() {
    }

    public int getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPass() {
        return pass;
    }

    public void setPass(String pass) {
        this.pass = pass;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", pass='" + pass + '\'' +
                '}';
    }
}
```

4. 创建spring配置文件 applicationContext.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"

       xsi:schemaLocation="
```

```
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.3.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

    <bean id="helloDao" class="com.hs.dao.impl.HelloDaoImpl">
    </bean>

    <!-- 设置注入 通过调用setter方法来实现注入 就必须有setter方法，否则无法注入-->
    <bean id="user" class="com.hs.pojo.User">
        <property name="id" value="2"/>
        <property name="name" value="李四"/>
        <property name="pass" value="111111"/>
    </bean>

    <!--构造注入 通过构造方法来实现注入 第一个参数索引是0 必须有构造方法-->
    <bean id="user2" class="com.hs.pojo.User">
        <constructor-arg index="0" value="3"/>
        <constructor-arg index="1" value="王五"/>
        <constructor-arg index="2" value="123456"/>
    </bean>

</beans>
```

5. 在src/test/java底下编写测试类Test.java

```
import com.hs.pojo.User;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test2 {

    @Test
    public void testUser1(){
        User user=new User(1,"张三","123456");
        System.out.println(user);
    }

    @Test
    public void testUser2(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        User user = (User) ctx.getBean("user");
        System.out.println(user);
    }
    @Test
    public void testUser3(){

        ApplicationContext ctx=new
```

```
ClassPathXmlApplicationContext("applicationContext.xml");
        User user = (User) ctx.getBean("user2");
        System.out.println(user);
    }
}
```

3. 自动装配

1. 创建maven工程（同上）
2. pom.xml文件里面导入依赖（同上）
3. 在src/main/java底下创建Student类，Clazz类

   Student.java:

```
import org.springframework.beans.factory.annotation.Autowired;

public class Student {

    int stuNo;
    String stuName;
    String address;

    Clazz clazz;//一对一关系


    // 省略gettersetter方法
}
```

   Clazz.java:

```

public class Clazz {

    private String className;
    private String address;

    //省略gettersetter方法
}
```

4. 编写applicationContext.xml配置文件

   手动装配：

```
    <bean id="clazz1" class="com.hs.pojo.Clazz">
        <property name="className" value="Java1班"/>
        <property name="address" value="101"/>
    </bean>

    <bean id="clazz" class="com.hs.pojo.Clazz">

        <property name="className" value="Java2班"/>
```

```
        <property name="address" value="102"/>
    </bean>



    <!-- 手动装配 -->
    <!-- 一对一关系使用ref属性 -->
    <bean id="student" class="com.hs.pojo.Student">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
        <property name="clazz" ref="clazz"/>
    </bean>
```

测试调用：

```
import com.hs.pojo.Student;
import com.hs.pojo.User;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test3 {

    @Test
    public void testStu1(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Student student = (Student) ctx.getBean("student");
        System.out.println(student);
    }
}
```

按照类型自动装配：

```
    <!--按照类型的自动装配必须 该种类型的对象只能有一个,所以Clazz对象需要注释掉一个-->
    <!--
    <bean id="clazz1" class="com.hs.pojo.Clazz">
        <property name="className" value="Java1班"/>
        <property name="address" value="101"/>
    </bean>-->

    <bean id="clazz" class="com.hs.pojo.Clazz">
        <property name="className" value="Java2班"/>
        <property name="address" value="102"/>
    </bean>
```

```
    <!-- 手动装配 -->
    <!-- 一对一关系使用ref属性 -->
    <bean id="student" class="com.hs.pojo.Student">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
        <property name="clazz" ref="clazz"/>
    </bean>

    <!-- 自动装配 autowire="" byType 按类型自动装配 只要属性的类型名称和待装配的类型保持一致，就自动装配，默认ByType-->
    <!--如果 byType时 同种类型的对象有多个时，不知道该装配那个，此时会报异常-->
    <bean id="student2" class="com.hs.pojo.Student" autowire="byType">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
    </bean>
```

按照类型的自动装配必须 该种类型的对象只能有一个；否则系统将会不知道装配那个，会报错。所以上面的Clazz对象需要注释掉一个

测试调用：

```
@Test
public void testStu2(){
    ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
    Student student = (Student) ctx.getBean("student2");
    System.out.println(student);
}
```

按照名称自动装配：

```
<!-- 自动装配 autowire="" byName 按名称自动装配 只要属性的名称和待装配的id值保持一致，就自动装配-->
    <bean id="student3" class="com.hs.pojo.Student" autowire="byName">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
    </bean>
```

测试调用：

```
@Test
public void testStu3(){
    ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
    Student student = (Student) ctx.getBean("student3");
    System.out.println(student);
}
```

通过注解方式实现自动装配：

需要在待装配的类上面加上注解@Autowired

```
package com.hs.pojo;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
public class Student {
```

```
    int stuNo;
    String stuName;
    String address;

    @Autowired
    Clazz clazz;//一对一关系

     // 省略gettersetter方法
}
```

需要有自动装配注解的支持的bean的配置

```
<!-- 注解方式实现自动装配-->
    <bean id="student4" class="com.hs.pojo.Student">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
    </bean>

    <bean
class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor">
</bean>
```

测试调用：

```
@Test
public void testStu4(){
    ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
    Student student = (Student) ctx.getBean("student4");
    System.out.println(student);
}
```

最终完整的代码：

application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.3.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

    <bean id="helloDao" class="com.hs.dao.impl.HelloDaoImpl">
    </bean>

    <!-- 设置注入 通过调用setter方法来实现注入 就必须有setter方法，否则无法注入-->
    <bean id="user" class="com.hs.pojo.User">
        <property name="id" value="2"/>
        <property name="name" value="李四"/>
        <property name="pass" value="111111"/>
    </bean>

    <!--构造注入 通过构造方法来实现注入 第一个参数索引是0 必须有构造方法-->
    <bean id="user2" class="com.hs.pojo.User">
        <constructor-arg index="0" value="3"/>
        <constructor-arg index="1" value="王五"/>
        <constructor-arg index="2" value="123456"/>
    </bean>
    <!--按照类型的自动装配必须 该种类型的对象只能有一个,所以Clazz对象需要注释掉一个-->
    <!--
    <bean id="clazz1" class="com.hs.pojo.Clazz">
        <property name="className" value="Java1班"/>
        <property name="address" value="101"/>
    </bean>-->

    <bean id="clazz" class="com.hs.pojo.Clazz">
        <property name="className" value="Java2班"/>

        <property name="address" value="102"/>
```

```
    </bean>



    <!-- 手动装配 -->
    <!-- 一对一关系使用ref属性 -->
    <bean id="student" class="com.hs.pojo.Student">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
        <property name="clazz" ref="clazz"/>
    </bean>

    <!-- 自动装配 autowire="" byType 按类型自动装配 只要属性的类型名称和待装配的类型保持一致，就自动装配，默认ByType-->
    <!--如果 byType时 同种类型的对象有多个时，不知道该装配那个，此时会报异常-->
    <bean id="student2" class="com.hs.pojo.Student" autowire="byType">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
    </bean>

    <!-- 自动装配 autowire="" byName 按名称自动装配 只要属性的名称和待装配的id值保持一致，就自动装配-->
    <bean id="student3" class="com.hs.pojo.Student" autowire="byName">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
    </bean>

    <!-- 注解方式实现自动装配-->
    <bean id="student4" class="com.hs.pojo.Student">
        <property name="stuNo" value="20201010"/>
        <property name="stuName" value="孟子"/>
        <property name="address" value="济南"/>
    </bean>

    <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor">
</bean>
</beans>
```

Test3.java

```
import com.hs.pojo.Student;
import com.hs.pojo.User;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test3 {

    @Test
```

```
    public void testStu1(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Student student = (Student) ctx.getBean("student");
        System.out.println(student);
    }

    @Test
    public void testStu2(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Student student = (Student) ctx.getBean("student2");
        System.out.println(student);
    }

    @Test
    public void testStu3(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Student student = (Student) ctx.getBean("student3");
        System.out.println(student);
    }

    @Test
    public void testStu4(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Student student = (Student) ctx.getBean("student4");
        System.out.println(student);
    }
}
```

4. 集合类型注入

1. 需要有maven工程
2. pom.xml文件里面需要引入相应的依赖
3. 编写实体类Order

   Order.java

```
package com.hs.pojo;

import java.util.*;

public class Order {
```

```
    private List<String> list=new ArrayList<>();
    private Set<Integer> set=new HashSet<>();
    private Properties properties=new Properties();
    private Map<Integer,String> map=new HashMap<>();

    //gettersetter方法省略
}
```

4. 在applicationContext.xml文件里面编写相应配置

```
    <!--注入各种集合类型-->
    <bean id="order" class="com.hs.pojo.Order">

        <!-- 注入List集合-->
        <property name="list">
            <list>
                <value>张三</value>
                <!-- 注入一个null值-->
                <null></null>
                <value>李四</value>
                <value>王五</value>
            </list>
        </property>

        <!-- 注入Set集合-->
        <property name="set">
            <set>
                <value>111</value>
                <value>222</value>
                <value>333</value>
            </set>
        </property>
        <!-- 注入Properties集合-->
        <property name="properties">
            <props>
                <prop key="aaa">111</prop>
                <prop key="bbb">222</prop>
                <prop key="ccc">333</prop>
            </props>
        </property>

        <!-- 注入Map集合-->
        <property name="map">
            <map>
                <entry key="10086" value="中国移动"></entry>
                <entry key="10010" value="中国联通"></entry>
                <entry key="10000" value="中国电信"></entry>
            </map>
        </property>
    </bean>
```

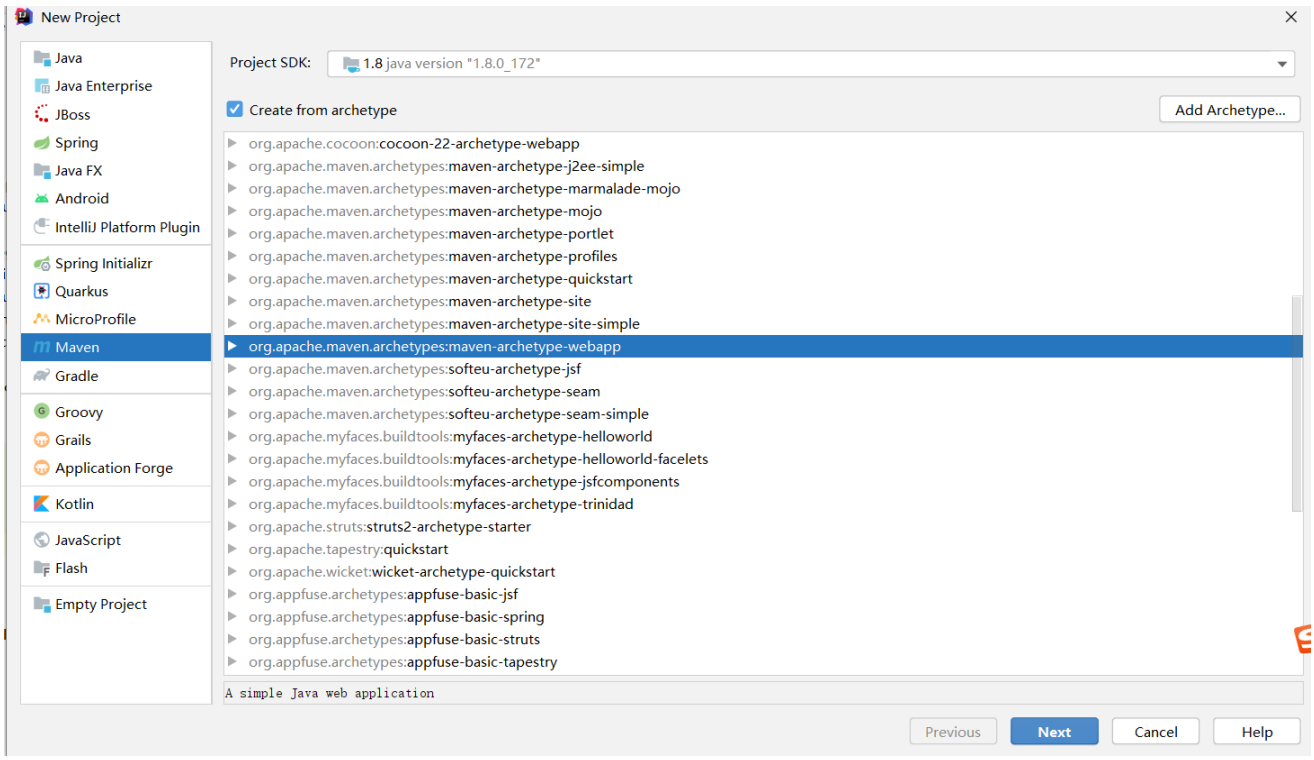
5. 编写测试类Test.java

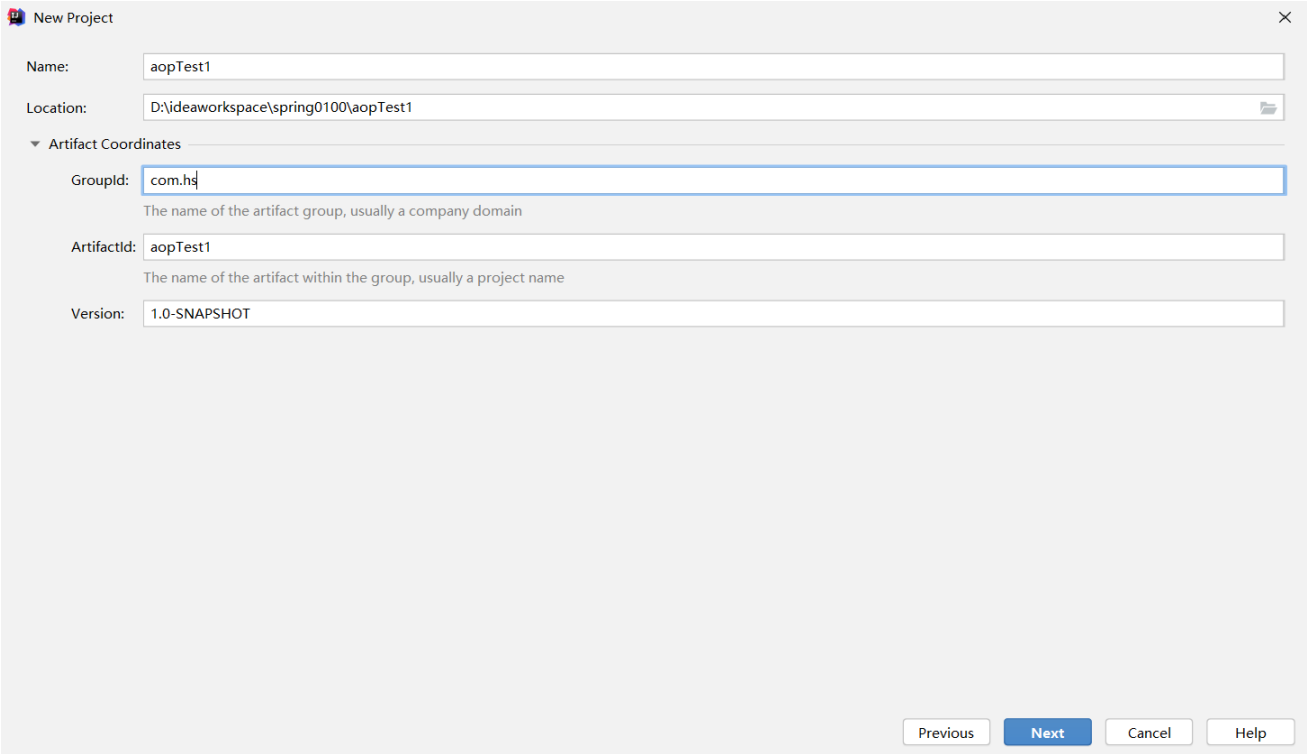
```
import com.hs.pojo.Order;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test4 {

    @Test
    public void testOrder(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Order order= (Order) ctx.getBean("order");
        System.out.println(order);
    }
}
```

aop笔记

1. 通过xml配置aop

1. 创建maven web工程





2. pom.xml文件导入依赖

```
 <!-- 从这开始 -->
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
  <!-- spring版本号 -->
  <spring.version>4.3.14.RELEASE</spring.version>

  <!-- log4j日志文件管理包版本 -->
  <slf4j.version>1.7.22</slf4j.version>
  <log4j.version>1.2.17</log4j.version>


</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>${log4j.version}</version>
  </dependency>
  <!-- spring -->
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
  <version>${spring.version}</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.8.13</version>
</dependency>



<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-expression</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-instrument</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-instrument-tomcat</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-messaging</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-oxm</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
```

```
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
  </dependency>


  <!-- mybatis 包 -->
  <dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.6</version>
  </dependency>

  <!--mybatis spring 插件 -->
  <dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.2</version>
  </dependency>

  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.45</version>
  </dependency>



  <!-- 上传下载 -->
  <dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.2</version>
  </dependency>
  <dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.2</version>
  </dependency>
  <dependency>
    <groupId>taglibs</groupId>
    <artifactId>standard</artifactId>
    <version>1.1.2</version>
  </dependency>

  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>8.0</version>
    <scope>provided</scope>
  </dependency>
```

```
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>8.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>net.sf.json-lib</groupId>
      <artifactId>json-lib</artifactId>
      <version>2.1</version>
      <classifier>jdk15</classifier>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>


    <!-- pagehelper分页插件 -->
    <!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper -->
    <dependency>
      <groupId>com.github.pagehelper</groupId>
      <artifactId>pagehelper</artifactId>
      <version>5.1.2</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.github.jsqlparser/jsqlparser -->
    <dependency>
      <groupId>com.github.jsqlparser</groupId>
      <artifactId>jsqlparser</artifactId>
      <version>0.9.5</version>
    </dependency>

    <!-- poi上传下载组件 -->
    <dependency>
      <groupId>org.apache.poi</groupId>
      <artifactId>poi-ooxml</artifactId>
      <version>3.14-beta1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.poi</groupId>
      <artifactId>poi-ooxml-schemas</artifactId>
      <version>3.14-beta1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.poi</groupId>
      <artifactId>poi</artifactId>
      <version>3.14-beta1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>4.5.2</version>
```

```
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.apache.xmlbeans/xmlbeans -->
    <dependency>
      <groupId>org.apache.xmlbeans</groupId>
      <artifactId>xmlbeans</artifactId>
      <version>2.6.0</version>
    </dependency>


    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>druid</artifactId>
      <version>1.1.10</version>
    </dependency>

  </dependencies>

   <!-- 到这结束 -->
```

3. 编写相应的javaBean类

```
package com.hs.pojo;

public class Printer {

    public void print(){
        System.out.println("打印机正在打印。。。。。");
    }
}
```

```
package com.hs.pojo;

public class Logger {

    public void loggerBefore(){
        System.out.println("方法开始执行了。。。。。");
    }

    public void loggerAfter(){
        System.out.println("方法执行完毕了。。。。。");
    }
}
```

```
package com.hs.pojo;

import java.util.Date;

public class Timer {

    Date date=new Date();
    public void showTime(){
        System.out.println("当前时间是:  "+date);
    }
}
```

4. 创建spring配置文件，编写相应配置

   applicationContext.xml

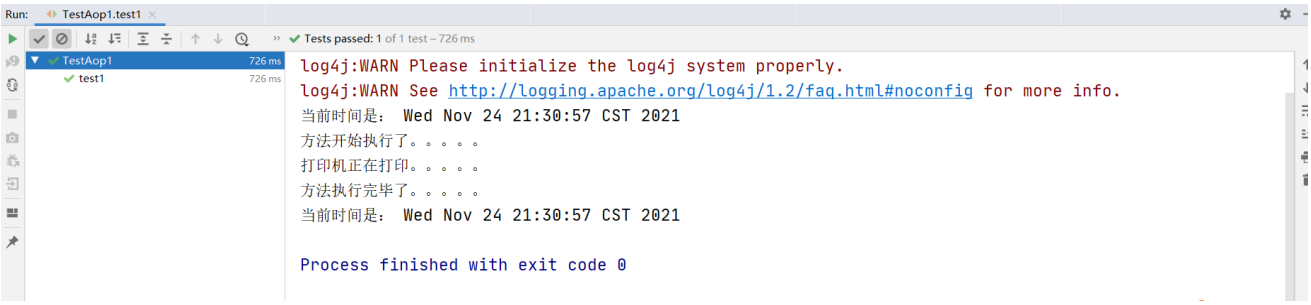
```
<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context-4.3.xsd
         http://www.springframework.org/schema/tx
         http://www.springframework.org/schema/tx/spring-tx-4.3.xsd
     http://www.springframework.org/schema/aop
         http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
    <!--基于xml方式来实现aop配置-->
    <bean id="p" class="com.hs.pojo.Printer"></bean>
    <bean id="log" class="com.hs.pojo.Logger"></bean>
    <bean id="time" class="com.hs.pojo.Timer"></bean>

    <!--aop配置-->
    <aop:config>

        <!--配置切入点-->
        <aop:pointcut id="addMethods" expression="execution(* com.hs.pojo.*.*(..) )"/>
        <!--配置切面1 order 优先级 值越小 优先级越高-->
        <aop:aspect id="t" ref="time" order="1">
            <aop:before method="showTime" pointcut-ref="addMethods"></aop:before>
            <aop:after method="showTime" pointcut-ref="addMethods"></aop:after>
        </aop:aspect>

        <!--配置切面2 order 优先级 值越小 优先级越高-->
        <aop:aspect id="l" ref="log" order="2">
            <aop:before method="loggerBefore" pointcut-ref="addMethods"></aop:before>
```

```
            <aop:after method="loggerAfter" pointcut-ref="addMethods"></aop:after>
        </aop:aspect>



    </aop:config>

</beans>
```

5. 测试相应代码， 单元测试

```
import com.hs.pojo.Printer;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestAop1 {

    @Test
    public void test1(){
        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Printer printer= (Printer) ctx.getBean("p");
        printer.print();
    }
}
```

最终执行效果：



2. 通过注解配置aop

1. 创建maven web工程 同上
2. 在pom.xml文件里面引入依赖 同上

3. 编写相应的javaBean类

```
package com.hs.pojo;

import org.springframework.stereotype.Component;

@Component
public class Printer {

    public void print(){
        System.out.println("打印机正在打印。。。。。");
    }
}
```

```
package com.hs.pojo;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
@Aspect
@Order(2)
public class Timer {

    Date date=new Date();

    @Around("execution(* com.hs.pojo.*.*(..))")
    public void showTime(ProceedingJoinPoint joinPoint) throws Throwable {
        System.out.println("开始时间是:  "+date);
        joinPoint.proceed();//在此方法上面的逻辑之前Before 之后的逻辑之后After
        System.out.println("结束时间是:  "+date);
    }
}
```

```
package com.hs.pojo;


import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.core.annotation.Order;
```

```
import org.springframework.stereotype.Component;

@Component
@Aspect
@Order(1)
public class Logger {

    @Before("execution(* com.hs.pojo.*.*(..))")
    public void loggerBefore(){
        System.out.println("方法开始执行了。。。。。");
    }

    @After("execution(* com.hs.pojo.*.*(..))")
    public void loggerAfter(){
        System.out.println("方法执行完毕了。。。。。");
    }
}
```

4. 创建spring配置文件，编写配置，并添加相应注解

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-4.3.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx-4.3.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

    <!--配置组件扫描器-->
    <context:component-scan base-package="com.hs"/>

    <!--配置aop相应的注解支持-->
    <aop:aspectj-autoproxy/>

</beans>
```

5. 测试相应代码， 单元测试

```
import com.hs.pojo.Printer;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestAop2 {

    @Test
    public  void test2(){

        ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Printer printer = ctx.getBean(Printer.class);
        printer.print();
    }
}
```

最终执行效果：

MyBatis笔记：

什么是MyBatis?

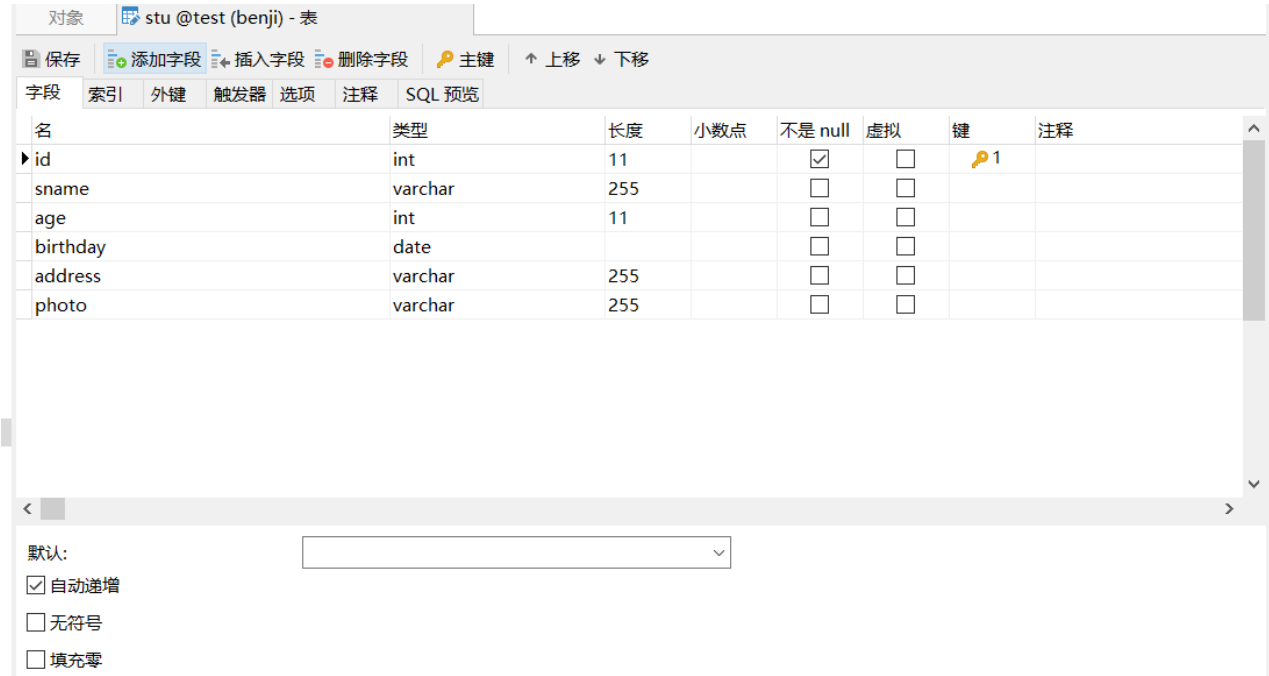
1. 基于java的数据库持久化框架。 使用java开发，做数据持久化。
2. 它是一个轻量级，半自动的orm持久化框架。一个轻量级，一个orm框架（Hibernate）数据库表和对象映射。
3. 通过映射xml来完成数据操作。在映射文件里面写sql语句

ORM: Object Relation Mapping 对象关系映射

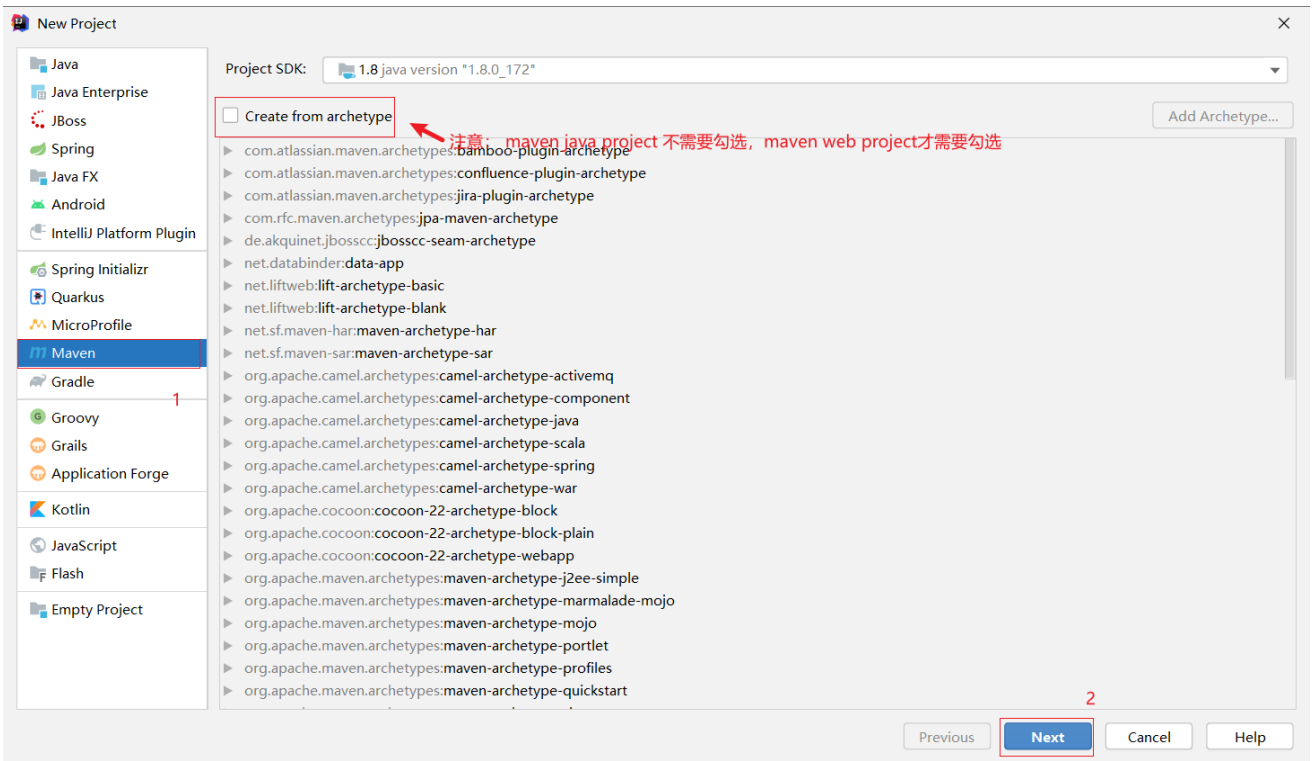
mybatis环境搭建：

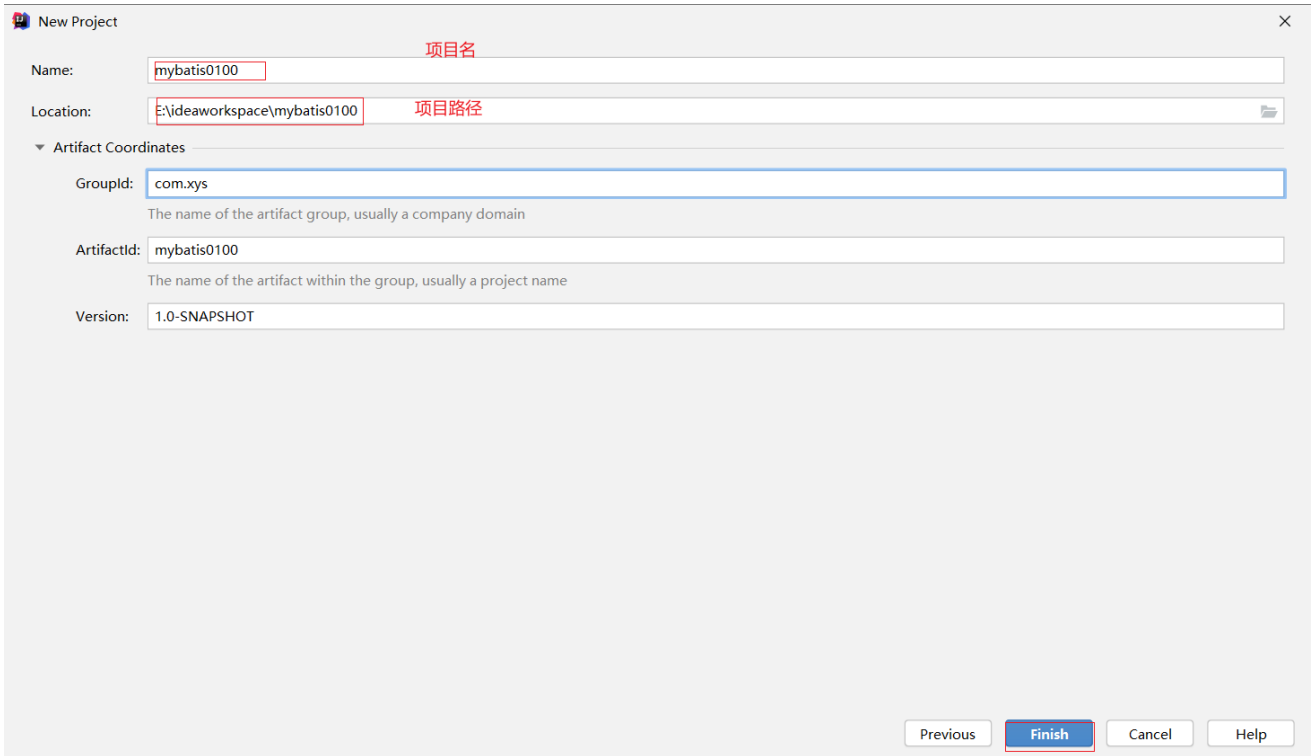
使用maven工程,mybatis是对jdbc的封装

0. 数据库表

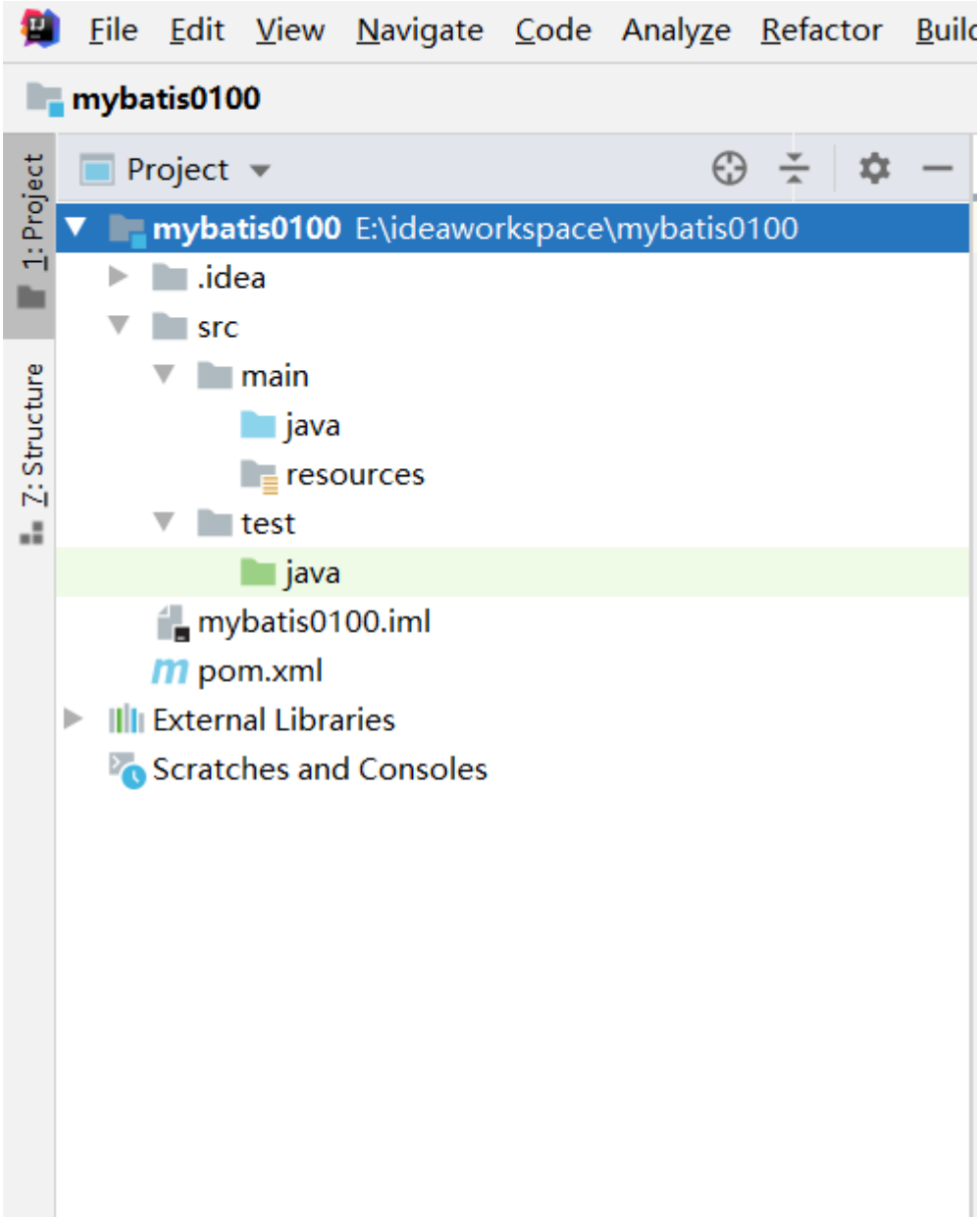


1. 创建maven project




New Project
项目名
Name:
mybatis0100
Location:
E:\ideaworkspace\mybatis0100
项目路径
Artifact Coordinates
GroupId:
com.xys
The name of the artifact group, usually a company domain
ArtifactId:
mybatis0100
The name of the artifact within the group, usually a project name
Version:
1.0-SNAPSHOT
Previous
Finish
Cancel
Help

项目结构如下：

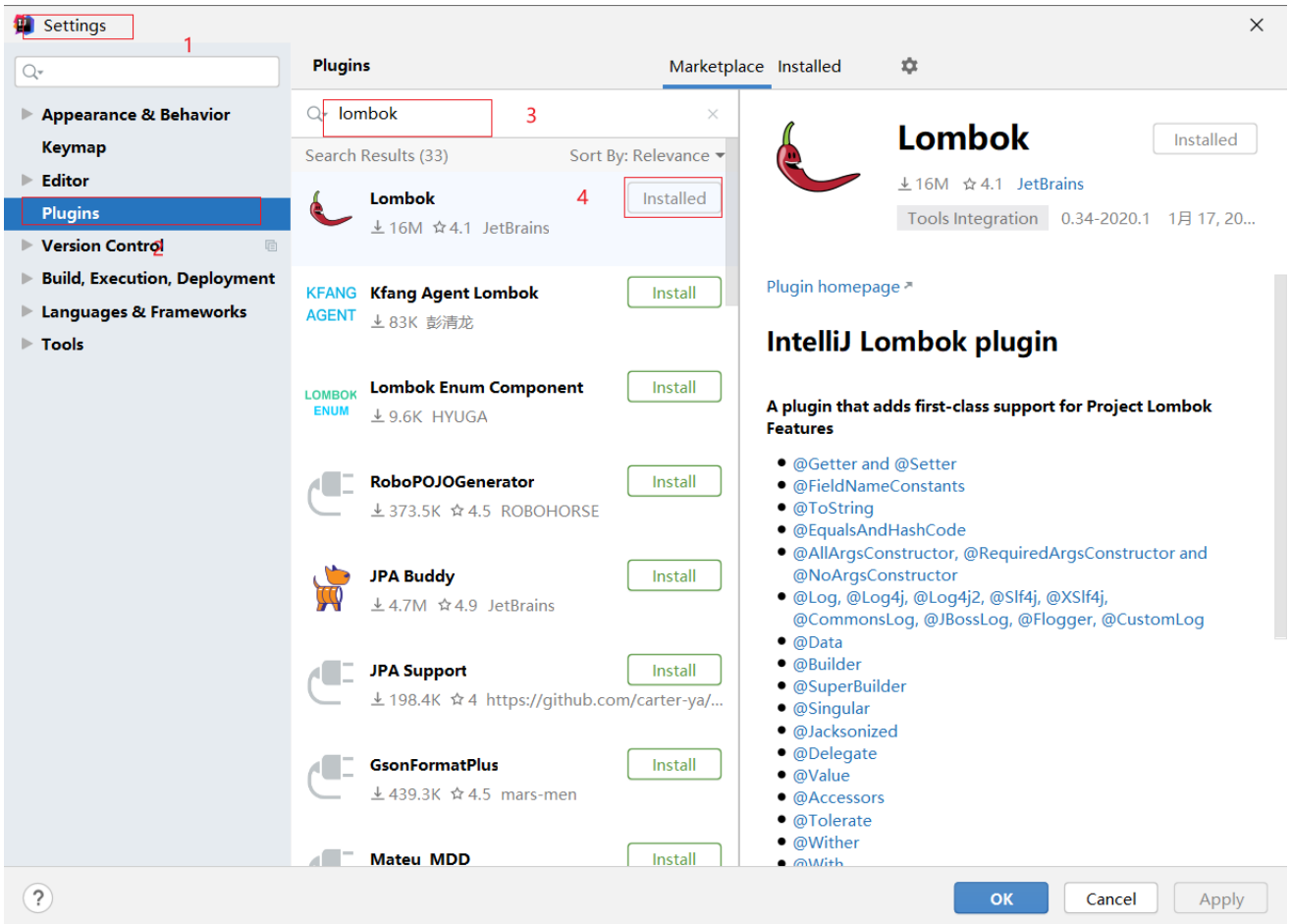


2. pom.xml 文件里面引入 mybatis和mysql依赖

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>

  <!--mybatis-->
```

```
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.4.6</version>
    </dependency>

    <!-- mysql-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.46</version>
    </dependency>

    <!--lombok-->
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <version>1.18.2</version>
    </dependency>
  </dependencies>
```

下载lombok插件: settings--plugins---搜索lombok插件---点击install



3. 创建实体类， dao接口

Student:

```
package com.hs.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;


@Data//相当于gettersetter方法和toString方法
@NoArgsConstructor//无参的构造方法
@AllArgsConstructor//有参的构造方法
public class Student {

    //目前属性名和字段名保持一致
    private int id;
    private String sname;
    private Date birthday;
    private int age;
    private String address;
    private String photo;
}
```

StudentDao:

```
package com.hs.dao;

import com.hs.pojo.Student;

public interface StudentDao {

    public Student queryOneById(int id);
}
```

4. 创建并配置mybatis配置文件

创建数据库连接的properties文件 db.properties，配置数据库连接相关信息

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/test?characterEncoding=utf-8&userSSL=false
jdbc.username=root
jdbc.password=123456
```

创建mybatis-config.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>

    <!--引入外部properties文件-->
    <properties resource="db.properties"/>
    <!-- 全局配置参数，需要时再设置 -->
    <!-- http://www.mybatis.org/mybatis-3/zh/configuration.html -->

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"></transactionManager>
            <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>

    <!--引入映射文件-->
    <mappers>
        <mapper resource="mapper/StudentMapper.xml"/>
    </mappers>

</configuration>
```

5. 创建并配置mybatis映射文件

StudentMapper.xml

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace="" 值必须和dao接口的路径保持一致-->
<mapper namespace="com.hs.dao.StudentDao">

    <!-- id值必须和接口中的方法名保持一致-->
    <select id="queryOneById" parameterType="int" resultType="com.hs.pojo.Student">
        select * from stu where id= #{id}
    </select>
</mapper>
```

6. 单元测试调用

```
import com.hs.pojo.Student;
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 org.junit.Test;

import java.io.IOException;
import java.io.InputStream;

public class TestMyBatis {


    @Test
    public void test01() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        SqlSession sqlSession = sqlSessionFactory.openSession();
        Student stu = sqlSession.selectOne("com.hs.dao.StudentDao.queryOneById", 26);
        System.out.println(stu);

    }
}
```

MyBatis配置文件：

类型别名（typeAliases）

类型别名可为 Java 类型设置一个缩写名字。 它仅用于 XML 配置，意在降低冗余的全限定类名书写。例如：

```
<typeAliases>
  <typeAlias alias="User" type="com.hs.pojo.User"/>
  <typeAlias alias="Student" type="com.hs.pojo.Student"/>
</typeAliases>
```

当这样配置时，User可以用在任何使用 com.hs.pojo.User 的地方。

也可以指定一个包名，MyBatis 会在包名下面搜索需要的 Java Bean，比如：

```
<typeAliases>
  <package name="com.hs.pojo"/>
</typeAliases>
```

Mappers 映射文件（映射文件路径）

Mapper配置的几种方法：

第一种（常用）

如：

第二种

使用完全限定路径

如：

第三种

使用mapper接口类路径

如：

注意：此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。

第四种（推荐）

注册指定包下的所有mapper接口

如：

注意：此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。

MyBatis配置日志显示SQL语句：

可以将mybatis执行过程中的sql语句直接在控制台上显示出来，需要相应配置：

1. pom.xml文件里面引入log4j的依赖

   pom.xml

```
<!-- log4j -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.9.1</version>
</dependency>
```

2. 在resource目录下创建log4j的属性配置文件 log4j.properties，放在resource下面的跟根目录，直接复制即可

   log4j.properties

```
log4j.rootLogger=debug,stdout,logfile
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout

log4j.logger.com.ibatis=DEBUG
log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG
log4j.logger.com.ibatis.common.jdbc.ScriptRunner=DEBUG
log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=DEBUG
log4j.logger.Java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
```

3. mybatis配置文件中 配置日志

   mybatis-config.xml

```
<settings>
       <setting name="logImpl" value="STDOUT_LOGGING"/>
</settings>
```

4. 单元测试，查看结果

   idea的控制台会显示sql语句及参数

```
Created connection 996796369.
==>  Preparing: select * from tb_user where username like "%武%"
==> Parameters:
<==    Columns: id, username, password, isadmin
<==        Row: 5, 武松, 7843784, 1
<==        Row: 6, 武松, 7843784, 2
<==      Total: 2
[User{id=5, name='武松', pass='7843784', isadmin=1}, User{id=6, name='武松',
pass='7843784', isadmin=2}]
Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@3b69e7d1]
Returned connection 996796369 to pool.
```

基于xml的映射和基于接口方式的映射：

StudentMapper.xml文件：

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace="" 值必须和dao接口的路径保持一致-->
<mapper namespace="com.hs.dao.StudentDao">

    <!-- id值必须和接口中的方法名保持一致-->
    <select id="queryOneById" parameterType="int" resultType="Student">
        select * from stu where id= #{id}
    </select>

    <!-- 查询所有-->
    <select id="queryAll" resultType="Student">
        select * from stu
    </select>

    <!--添加-->
    <insert id="insertStudent" parameterType="Student">
        INSERT INTO `stu` (`id`, `sname`, `age`, `birthday`, `address`, `photo`) VALUES (0,
#{sname}, #{age}, #{birthday}, #{address}, #{photo})
    </insert>

    <!--修改-->
    <update id="updateStudent" parameterType="Student">
        UPDATE `stu` SET  `sname`=#{sname}, `age`=#{age}, `birthday`=#{birthday},
`address`=#{address}, `photo`=#{photo} WHERE (`id`=#{id})
    </update>

    <!--删除-->
```

```
    <delete id="deleteStudentById" parameterType="int">
        delete from stu where id=#{id}
    </delete>

    <!--模糊查询-->

    <select id="queryByMohu" resultType="Student" parameterType="java.lang.String" >
        select * from stu where sname like "%${mohu}%"
    </select>
```

基于xml方式映射：

直接定位到xml映射文件里面的某个标签

TestMyBatis.java

```
import com.hs.pojo.Student;
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 org.junit.Test;

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

public class TestMyBatis {


    @Test
    public void test01() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除
        Student stu = sqlSession.selectOne("com.hs.dao.StudentDao.queryOneById", 26);
        System.out.println(stu);
    }

    @Test
    public void testQueryAll() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
```

```
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除
        List<Student> list = sqlSession.selectList("mapper.StudentMapper.queryAll");
        System.out.println(list);
    }

    @Test
    public void testInsert() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        //参数true表示会默认自动提交事务
        SqlSession sqlSession = sqlSessionFactory.openSession(true);
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除

        Student student=new Student(0,"aaa",new Date(),23,"aaa","aaa");
        int row = sqlSession.insert("com.hs.dao.StudentDao.insertStudent", student);
        //sqlSession.commit();//手动提交
        System.out.println("添加了"+row+"行");
        sqlSession.close();
    }
    @Test
    public void testUpdate() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        //参数true表示会默认自动提交事务
        SqlSession sqlSession = sqlSessionFactory.openSession(true);
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除

        Student student=new Student(232,"bbb",new Date(),32,"bbb","bbb");
        sqlSession.update("com.hs.dao.StudentDao.updateStudent",student);
        sqlSession.close();
    }

    @Test
    public void testDelete() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        //参数true表示会默认自动提交事务
        SqlSession sqlSession = sqlSessionFactory.openSession(true);
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除
```

```
        int row = sqlSession.delete("com.hs.dao.StudentDao.deleteStudentById", 232);
        System.out.println("删除了"+row+"行");
        sqlSession.close();
    }

    @Test
    public void testQueryByMohu() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除

        List<Object> list = sqlSession.selectList("mapper.StudentMapper.queryByMohu","钱");
        System.out.println(list);
        sqlSession.close();
    }
}
```

基于接口方式来映射：

StudentDao.java

```
package com.hs.dao;

import com.hs.pojo.Student;
import org.apache.ibatis.annotations.Param;

import java.util.List;

public interface StudentDao {

    //查询单个
    public Student queryOneById(int id);

    //查询所有
    public List<Student> queryAll();

    //添加
    public int insertStudent(Student student);

    //修改
    public int updateStudent(Student student);

    //删除
    public int deleteStudentById(int id);
```

```
    //模糊查询
    public List<Student> queryByMohu(@Param("mohu") String mohu);
}
```

StudentMapper.xml

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace="" 值必须和dao接口的路径保持一致-->
<mapper namespace="com.hs.dao.StudentDao">

    <!-- id值必须和接口中的方法名保持一致-->
    <select id="queryOneById" parameterType="int" resultType="Student">
        select * from stu where id= #{id}
    </select>

    <!-- 查询所有-->
    <select id="queryAll" resultType="Student">
        select * from stu
    </select>

    <!--添加-->
    <insert id="insertStudent" parameterType="Student">
        INSERT INTO `stu` (`id`, `sname`, `age`, `birthday`, `address`, `photo`) VALUES (0,
#{sname}, #{age}, #{birthday}, #{address}, #{photo})
    </insert>

    <!--修改-->
    <update id="updateStudent" parameterType="Student">
        UPDATE `stu` SET  `sname`=#{sname}, `age`=#{age}, `birthday`=#{birthday},
`address`=#{address}, `photo`=#{photo} WHERE (`id`=#{id})
    </update>

    <!--删除-->
    <delete id="deleteStudentById" parameterType="int">
        delete from stu where id=#{id}
    </delete>

    <!--模糊查询-->

    <select id="queryByMohu" resultType="Student" parameterType="java.lang.String" >
        select * from stu where sname like "%${mohu}%"
    </select>



</mapper>
```

TestMapper.java

```
import com.hs.dao.StudentDao;
import com.hs.pojo.Student;
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 org.junit.After;
import org.junit.Before;
import org.junit.Test;

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

public class TestMapper {

    private SqlSession sqlSession;

    @Before
    public void init() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        sqlSession = sqlSessionFactory.openSession(true);
    }

    @After
    public void close(){
        sqlSession.close();
    }

    @Test
    public void testQueryAll(){

       //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Student> list = studentDao.queryAll();
        System.out.println(list);
    }

    @Test
    public void testQueryOne(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        Student student = studentDao.queryOneById(30);
        System.out.println(student);
    }
```

```
    @Test
    public void testAddOne(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        Student student=new Student(0,"aaa",new Date(),23,"aaa","aaa");
        int row = studentDao.insertStudent(student);
        System.out.println("添加了"+row+"行");
    }

    @Test
    public void testUpdateOne(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        Student student=new Student(233,"ccc",new Date(),23,"ccc","ccc");
        int row = studentDao.updateStudent(student);
        System.out.println("修改了"+row+"行");
    }

    @Test
    public void testDelete(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        int row = studentDao.deleteStudentById(233);
        System.out.println("删除了"+row+"行");
    }

    @Test
    public void testQueryByMohu(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Student> list = studentDao.queryByMohu("钱");
        System.out.println(list);
    }
}
```

基于注解方式的映射：

吧映射文件可以直接删掉

StudentDao.java:

```
package com.hs.dao;

import com.hs.pojo.Student;
import org.apache.ibatis.annotations.*;
```

```
import java.util.List;

public interface StudentDao {

    //查询单个
    @Select(" select * from stu where id= #{id}")
    public Student queryOneById(int id);

    //查询所有
    @Select(" select * from stu")
    public List<Student> queryAll();

    //添加
    @Insert("INSERT INTO `stu` (`id`, `sname`, `age`, `birthday`, `address`, `photo`)
VALUES (0, #{sname}, #{age}, #{birthday}, #{address}, #{photo})")
    public int insertStudent(Student student);

    //修改
    @Update(" UPDATE `stu` SET  `sname`=#{sname}, `age`=#{age}, `birthday`=#{birthday},
`address`=#{address}, `photo`=#{photo} WHERE (`id`=#{id})")
    public int updateStudent(Student student);

    //删除
    @Delete(" delete from stu where id=#{id}")
    public int deleteStudentById(int id);

    //模糊查询
    @Select("select * from stu where sname like '%${mohu}%'")
    public List<Student> queryByMohu(@Param("mohu") String mohu);
}
```

调用方式和接口方式是一样的：

```
import com.hs.dao.StudentDao;
import com.hs.pojo.Student;
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 org.junit.After;
import org.junit.Before;
import org.junit.Test;

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

public class TestMapper {
```

```
    private SqlSession sqlSession;

    @Before
    public void init() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        sqlSession = sqlSessionFactory.openSession(true);
    }

    @After
    public void close(){
        sqlSession.close();
    }

    @Test
    public void testQueryAll(){

       //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Student> list = studentDao.queryAll();
        System.out.println(list);
    }

    @Test
    public void testQueryOne(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        Student student = studentDao.queryOneById(30);
        System.out.println(student);
    }

    @Test
    public void testAddOne(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        Student student=new Student(0,"aaa",new Date(),23,"aaa","aaa");
        int row = studentDao.insertStudent(student);
        System.out.println("添加了"+row+"行");
    }

    @Test
    public void testUpdateOne(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        Student student=new Student(233,"ccc",new Date(),23,"ccc","ccc");
        int row = studentDao.updateStudent(student);
        System.out.println("修改了"+row+"行");
    }
```

```
    @Test
    public void testDelete(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        int row = studentDao.deleteStudentById(233);
        System.out.println("删除了"+row+"行");
    }

    @Test
    public void testQueryByMohu(){

        //基于接口方式来映射
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Student> list = studentDao.queryByMohu("钱");
        System.out.println(list);
    }
}
```

mybatis单表增删改查：

1. 工程结构，配置信息同上
2. StudentDao接口里面增删改查方法

   StudentDao:

```
package com.hs.dao;

import com.hs.pojo.Student;
import org.apache.ibatis.annotations.Param;

import java.util.List;

public interface StudentDao {

    //查询单个
    public Student queryOneById(int id);

    //查询所有
    public List<Student> queryAll();

    //添加
    public int insertStudent(Student student);

    //修改
```

```
    public int updateStudent(Student student);

    //删除
    public int deleteStudentById(int id);

    //模糊查询
    public List<Student> queryByMohu(@Param("mohu") String mohu);
}
```

3. StudentMapper.xml映射文件里面配置增删改查映射

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace="" 值必须和dao接口的路径保持一致-->
<mapper namespace="com.hs.dao.StudentDao">

    <!-- id值必须和接口中的方法名保持一致-->
    <select id="queryOneById" parameterType="int" resultType="Student">
        select * from stu where id= #{id}
    </select>

    <!-- 查询所有-->
    <select id="queryAll" resultType="Student">
        select * from stu
    </select>

    <!--添加-->
    <insert id="insertStudent" parameterType="Student">
        INSERT INTO `stu` (`id`, `sname`, `age`, `birthday`, `address`, `photo`) VALUES
(0, #{sname}, #{age}, #{birthday}, #{address}, #{photo})
    </insert>

    <!--修改-->
    <update id="updateStudent" parameterType="Student">
        UPDATE `stu` SET  `sname`=#{sname}, `age`=#{age}, `birthday`=#{birthday},
`address`=#{address}, `photo`=#{photo} WHERE (`id`=#{id})
    </update>

    <!--删除-->
    <delete id="deleteStudentById" parameterType="int">
        delete from stu where id=#{id}
    </delete>

    <!--模糊查询-->

    <select id="queryByMohu" resultType="Student" parameterType="java.lang.String" >
        select * from stu where sname like "%${mohu}%"
    </select>



</mapper>
```

4. 单元测试测试调用

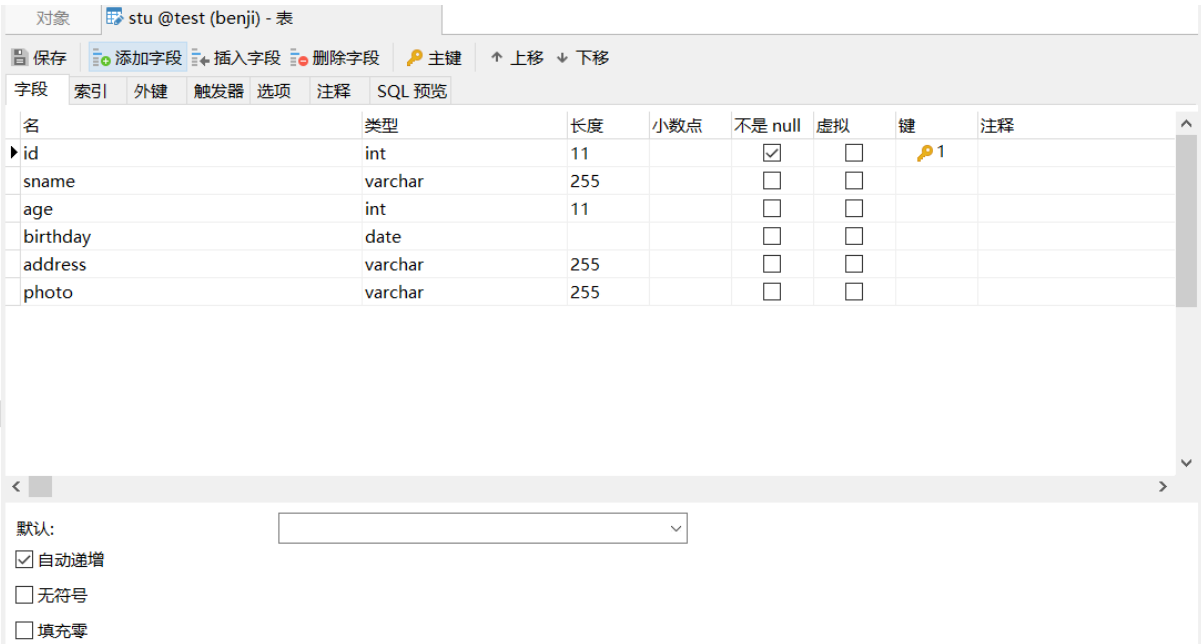
```
import com.hs.pojo.Student;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Test;

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

public class TestMyBatis {


    @Test
    public void test01() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除
        Student stu = sqlSession.selectOne("com.hs.dao.StudentDao.queryOneById", 26);
        System.out.println(stu);
    }

    @Test
    public void testQueryAll() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除
        List<Student> list = sqlSession.selectList("mapper.StudentMapper.queryAll");
        System.out.println(list);
    }

    @Test
    public void testInsert() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        //参数true表示会默认自动提交事务
```

```
        SqlSession sqlSession = sqlSessionFactory.openSession(true);
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除

        Student student=new Student(0,"aaa",new Date(),23,"aaa","aaa");
        int row = sqlSession.insert("com.hs.dao.StudentDao.insertStudent", student);
        //sqlSession.commit();//手动提交
        System.out.println("添加了"+row+"行");
        sqlSession.close();
    }
    @Test
    public void testUpdate() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        //参数true表示会默认自动提交事务
        SqlSession sqlSession = sqlSessionFactory.openSession(true);
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除

        Student student=new Student(232,"bbb",new Date(),32,"bbb","bbb");
        sqlSession.update("com.hs.dao.StudentDao.updateStudent",student);
        sqlSession.close();
    }

    @Test
    public void testDelete() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        //参数true表示会默认自动提交事务
        SqlSession sqlSession = sqlSessionFactory.openSession(true);
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除

        int row = sqlSession.delete("com.hs.dao.StudentDao.deleteStudentById", 232);
        System.out.println("删除了"+row+"行");
        sqlSession.close();
    }

    @Test
    public void testQueryByMohu() throws IOException {
        String resource="mybatis-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //SqlSession  表示Sql会话，某个用户与数据库建立连接之后的多个sql操作
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //selectOne 查询单个 第一个参数写成namespace.id值
        //selectList()  查询所有 insert()添加 update() 修改 delete() 删除

        List<Object> list =
```

```
sqlSession.selectList("mapper.StudentMapper.queryByMohu","钱");
        System.out.println(list);
        sqlSession.close();
    }
}
```

使用ResultMap标签解决属性名和字段名冲突不一致的情形：

1. 创建数据库表tb_stu表：



2. Student实体类

```
package com.hs.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;


@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {

    private int id;
    private String stuName;
    private int age;
    private Date birth;
    private String address;
```

```
    private String image;
}
```

3. StudentDao接口

```
package com.hs.dao;

import com.hs.pojo.Student;
import com.hs.pojo.Teacher;

import java.util.List;

public interface StudentDao {

    public List<Student> queryAll();

    public List<Student> queryAll2();


}
```

4. StudentDao.xml映射文件

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.hs.dao.StudentDao">

    <!--解决属性名和字段名冲突不一致的解决方案-->
    <!-- 一 ： 在sql语句里面给字段名起别名 -->
    <select id="queryAll" resultType="Student">
        select id, sname stuName, age, birthday birth, address, photo image from stu
    </select>

    <!--通过resultMap 建立 属性与字段之间的映射关系-->
    <resultMap id="stuResultMap" type="Student">
        <!-- id 主键  result 简单类型 ： 基本数据类型和String类型 association 值得是 类类型 一对一关系 collection集合 一对多-->
        <id property="id"  column="id"/>
        <result property="stuName" column="sname"/>
        <result property="age" column="age"/>
        <result property="birth" column="birthday"/>
        <result property="address" column="address"/>
        <result property="image" column="photo"/>

    </resultMap>


    <select id="queryAll2" resultMap="stuResultMap">
        select * from stu
```

```
    </select>

</mapper>
```

5. mybatis-config.xml配置文件中引入映射文件

```
<!--引入映射文件-->
    <mappers>
        <!--<mapper resource="mapper/StudentDao.xml"/>-->
        <!-- 如果使用class属性要求xml文件名称和dao接口路径名称相同，并且文件名相同-->
        <mapper class="com.hs.dao.StudentDao"/>
    </mappers>
```

6. 单元测试

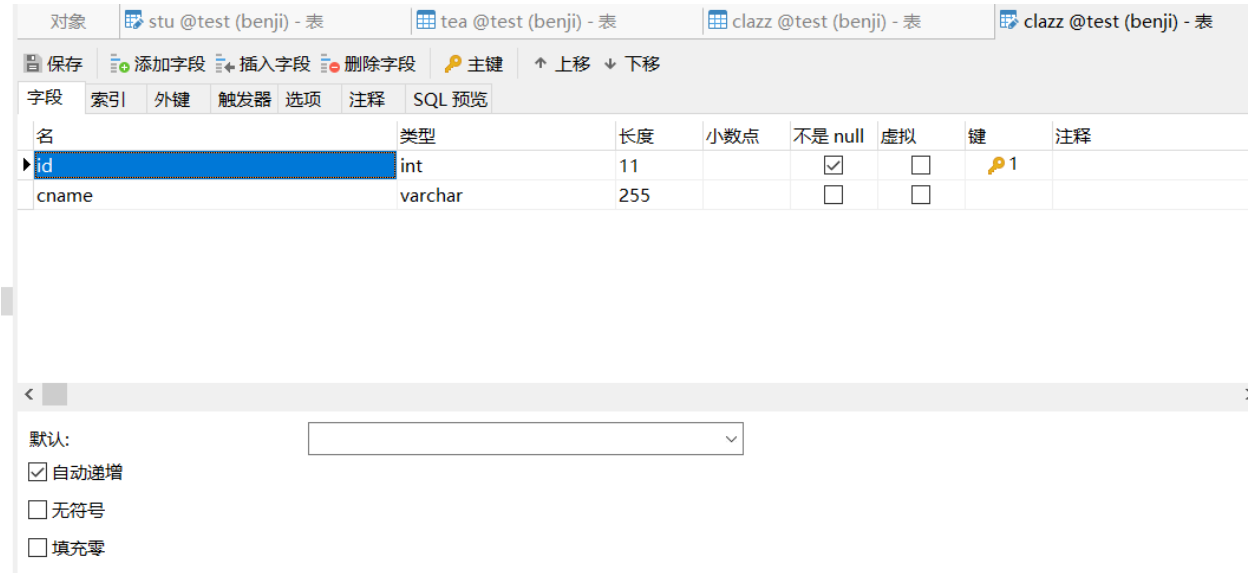
```
package test;

import com.hs.dao.StudentDao;
import com.hs.pojo.Student;
import com.hs.pojo.Teacher;
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 org.junit.After;
import org.junit.Before;
import org.junit.Test;

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

public class TestStu {

    private SqlSession sqlSession;

    @Before
    public void init() throws IOException {
        String resource="mybais-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //true 表示自动提交事务
        sqlSession = sqlSessionFactory.openSession(true);

    }

    @Test
    public void queryAll(){
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Student> list = studentDao.queryAll();
```

```
        System.out.println(list);
    }
    @After
    public void close(){
        sqlSession.close();
    }

    @Test
    public void queryAll2(){
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Student> list = studentDao.queryAll2();
        System.out.println(list);
    }


}
```

一对一关系和一对多关系：

通过resultMap标签中的association属性来配置

1. 数据库表 stu 学生表 和 clazz 班级表



对象 | stu @test (benji) ... | tea @test (benji)... | clazz @test (benj... | clazz @t

开始事务 文本 筛选 排序 导入 导出

id	cname
1	18软1
2	18软2
3	18软3
4	18软4

对象 | stu @test (benji) ... | tea @test (benji)... | clazz @test (benj... | clazz @test (benj... | stu @test (benji) ... | tea @test (benji)...

保存 添加字段 插入字段 删除字段 主键 上移 下移

字段 索引 外键 触发器 选项 注释 SQL 预览

名	类型	长度	小数点	不是 null	虚拟	键	注释
id	int	11		☑	☐	1	
tname	varchar	255		☐	☐		
age	int	11		☐	☐		
subject	varchar	255		☐	☐		
cid	int	11		☐	☐		

对象 | stu @test (benji) ... | tea @test (benji)... | clazz @test (b

开始事务 文本 筛选 排序 导入 导出

id	tname	age	subject	cid
1	孔子	23	java	1
2	孟子	26	c++	2
3	韩非子	56	法律	3
4	庄子	23	大数据	4
5	孙子	24	军事	1

2. 实体类 Student 和 Clazz 类

   Student:

```
package com.hs.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;


@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {
```

```
    private int id;
    private String stuName;
    private int age;
    private Date birth;
    private String address;
    private String image;
}
```

Clazz:

```
package com.hs.pojo;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Clazz {

    private int classId;
    private String className;
}
```

Teacher:

```
package com.hs.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Teacher {
    private int id;
    private String teaName;
    private int age;
    private String subject;
    private Clazz clazz;//老师和班级一对一
    private List<Student> students;//老师和学生 之间 一对多关系
```

```
}
```

3. 接口方法

StudentDao:

```
 //连表查询
    public List<Teacher> queryAllTeachers();
```

4. 映射文件配置

```
        <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.hs.dao.StudentDao">



    <!--通过resultMap 建立 属性与字段之间的映射关系-->
    <resultMap id="stuResultMap" type="Student">
        <!-- id 主键  result 简单类型 ：基本数据类型和String类型 association 值得是 类类型 一对一关系 collection集合 一对多-->
        <id property="id"  column="id"/>
        <result property="stuName" column="sname"/>
        <result property="age" column="age"/>
        <result property="birth" column="birthday"/>
        <result property="address" column="address"/>
        <result property="image" column="photo"/>

    </resultMap>

    <resultMap id="clazzResultMap" type="Clazz">
        <id property="classId" column="cid"/>
        <result property="className" column="cname"/>
    </resultMap>

    <resultMap id="teaResultMap" type="Teacher">
        <id property="id" column="tid"/>
        <result property="teaName" column="tname"/>
        <result property="age" column="age"/>
        <result property="subject" column="subject"/>
        <!--association 一对一关系-->
        <association property="clazz" resultMap="clazzResultMap"></association>
        <!--collection 一对多关系-->
        <collection property="students" ofType="Student" resultMap="stuResultMap">
</collection>
    </resultMap>



    <select id="queryAllTeachers" resultMap="teaResultMap">
        select t.*,c.*,s.*from stu s,clazz c ,tea t WHERE t.cid=c.cid and t.tid=s.tid
    </select>
```

```
</mapper>
```

5. 单元测试

```
 package test;

import com.hs.dao.StudentDao;
import com.hs.pojo.Student;
import com.hs.pojo.Teacher;
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 org.junit.After;
import org.junit.Before;
import org.junit.Test;

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

public class TestStu {

    private SqlSession sqlSession;

    @Before
    public void init() throws IOException {
        String resource="mybais-config.xml";
        InputStream in = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //true 表示自动提交事务
        sqlSession = sqlSessionFactory.openSession(true);

    }

    @Test
    public void queryAll(){
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Student> list = studentDao.queryAll();
        System.out.println(list);
    }
    @After
    public void close(){
        sqlSession.close();
    }


    @Test
    public void queryAllTeahers(){
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Teacher> list = studentDao.queryAllTeachers();
```

```
        System.out.println(list);
    }
}
```

动态SQL:

1. if test 判断

```
if test 就是if判断 test为判断的条件
```

2. where if test where标签

```
where 标签相当于sql语句中的where关键字，但能去除第一个多余的and 或者 or
```

3. trim if test 自定义标签 ， 可以实现 where或set的效果

```
trim为自定义标签 prefix前缀是什么 trim就相当于哪个标签的作用 prefixOverrides 值为and 或者 or
```

4. set if test 修改

```
set 标签相当于sql语句中的set关键字，但能去除最后一个多余的，
```

5. foreach 循环

```
foreach 循环
    属性:
    item:  集合中的每个元素
    index: 下标
    collection:  集合 值为 list/array/map
    separator: 每个元素的分隔符
    open: 整个循环以什么开头
    close: 整个循环以什么结尾
```

例子:

UserDao接口:

```
package com.hs.dao;

import com.hs.pojo.User;
import org.apache.ibatis.annotations.Param;

import java.util.List;


public interface UserDao {
    //模糊查询
    public List<User> queryStudentsByMohu(@Param("mohu") String mohu);
    //模糊查询
    public List<User> queryStudentsByMohu2(@Param("mohu") String mohu);
    //查询 直接查询
    public List<User> queryStudentsByUsernameAndAdmin(User user);
    //查询 属性不为空根聚这个属性查询，为空则不根据这个属性查询 if test
    public List<User> queryStudentsByUsernameAndAdmin2(User user);
    //查询 属性不为空根聚这个属性查询，为空则不根据这个属性查询 where if test
    public List<User> queryStudentsByUsernameAndAdmin3(User user);
    //查询 属性不为空根聚这个属性查询，为空则不根据这个属性查询 trim if test
    public List<User> queryStudentsByUsernameAndAdmin4(User user);
    //更新 属性不为空则更新，为空则不更新这个字段 set if test
    public void updateUser(User user);
    //根据id的集合查询所有
    public List<User> queryStudentsByIds(List<Long> ids);
    //批量添加
    public void insertBatch(List<User> users);

}
```

UserMapper.xml

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace 值必须和接口路径保持一致-->
<mapper namespace="com.hs.dao.UserDao">

    <!-- ResultMap是属性名和字段名冲突-->

    <resultMap id="userResultMap" type="User">

        <!-- id表示主键  result普通属性 association javabean collection 集合-->
        <id property="id" column="id"/>
        <result property="name" column="username"/>
        <result property="pass" column="password"/>
        <result property="isadmin" column="isadmin"/>

    </resultMap>
```

```
    <select id="queryStudentsByMohu" parameterType="string" resultMap="userResultMap">
        select * from tb_user where username like "%${mohu}%"
    </select>

    <select id="queryStudentsByMohu2" parameterType="string" resultMap="userResultMap">
        select * from tb_user where username like "%${mohu}%" or password like "%${mohu}%"
    </select>

    <select id="queryStudentsByUsernameAndAdmin" parameterType="User"
resultMap="userResultMap">
        select * from tb_user where username=#{name} and isadmin=#{isadmin}
    </select>

    <!--if test 就是if判断 test为判断的条件-->
    <select id="queryStudentsByUsernameAndAdmin2" parameterType="User"
resultMap="userResultMap">
        select * from tb_user where 1=1
        <if test="name!=null and name!=''">
            and username=#{name}
        </if>
        <if test="isadmin!=null and isadmin!=0">
            and isadmin=#{isadmin}
        </if>

    </select>

    <!-- where 标签相当于sql语句中的where关键字，但能去除第一个多余的and 或者 or-->
    <select id="queryStudentsByUsernameAndAdmin3" parameterType="User"
resultMap="userResultMap">
        select * from tb_user
        <where>
            <if test="name!=null and name!=''">
                and username=#{name}
            </if>
            <if test="isadmin!=null and isadmin!=0">
                and isadmin=#{isadmin}
            </if>
        </where>

    </select>

    <!-- trim为自定义标签 prefix前缀是什么 trim就相当于哪个标签的作用 prefixOverrides 值为and 或者
or-->
    <select id="queryStudentsByUsernameAndAdmin4" parameterType="User"
resultMap="userResultMap">
        select * from tb_user
        <trim prefix="where" prefixOverrides="and">
            <if test="name!=null and name!=''">
                and username=#{name}
            </if>
            <if test="isadmin!=null and isadmin!=0">
```

```
                and isadmin=#{isadmin}
            </if>
        </trim>
    </select>

    <!-- set 标签相当于sql语句中的set关键字，但能去除最后一个多余的,-->
    <update id="updateUser" parameterType="User">

        update tb_user
        <set>

            <if test="name!=null and name!=''">
                 username=#{name},
            </if>
            <if test="pass!=null and pass!=''">
                 password=#{pass},
            </if>
            <if test="isadmin!=null and isadmin!=0">
                 isadmin=#{isadmin}
            </if>

        </set>
        where id=#{id}
    </update>

    <!-- foreach 循环
        属性:
        item:  集合中的每个元素
        index: 下标
        collection:  集合 值为 list/array/map
        separator: 每个元素的分隔符
        open: 整个循环以什么开头
        close: 整个循环以什么结尾
    -->
    <select id="queryStudentsByIds" parameterType="List" resultMap="userResultMap">
         select * from tb_user where id in
         <foreach collection="list" item="id" index="i" open="(" separator="," close=")">
             #{id}
         </foreach>

    </select>

    <insert id="insertBatch" parameterType="List">
        INSERT INTO tb_user ( username, password, isadmin) VALUES
        <foreach collection="list" item="item" index="index" open="" close=""
separator=",">
            (#{item.name}, #{item.pass}, #{item.isadmin})
        </foreach>
    </insert>

</mapper>
```

TestUser： 测试类

```
import com.hs.dao.UserDao;
import com.hs.pojo.User;
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 org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
import java.io.InputStream;
import java.sql.SQLOutput;
import java.util.ArrayList;
import java.util.List;

public class TestUser {

    SqlSession session;

    @Before
    public  void getSession() throws IOException {
        //加载配置文件 并读取  通过IO流
        //1. 获取输入流
        InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        session = sqlSessionFactory.openSession(true);
    }

    @After
    public void close(){
        session.close();
    }

    @Test
    public void testQueryByMohu(){

        UserDao userDao = session.getMapper(UserDao.class);
        List<User> users = userDao.queryStudentsByMohu("武");
        System.out.println(users);
    }
    @Test
    public void testQueryByMohu2(){

        UserDao userDao = session.getMapper(UserDao.class);
        List<User> users = userDao.queryStudentsByMohu2("2");
        System.out.println(users);
    }
```

```
    @Test
    public void testQueryStudentsByUsernameAndAdmin(){

        UserDao userDao = session.getMapper(UserDao.class);
        User user=new User();
        user.setName("武松");
        user.setIsadmin(2);
        List<User> users = userDao.queryStudentsByUsernameAndAdmin(user);
        System.out.println(users);
    }

    @Test
    public void testQueryStudentsByUsernameAndAdmin2(){

        UserDao userDao = session.getMapper(UserDao.class);
        User user=new User();
        //user.setName("武松");
        user.setIsadmin(2);
        List<User> users = userDao.queryStudentsByUsernameAndAdmin2(user);
        System.out.println(users);
    }

    @Test
    public void testQueryStudentsByUsernameAndAdmin3(){

        UserDao userDao = session.getMapper(UserDao.class);
        User user=new User();
        //user.setName("武松");
        user.setIsadmin(2);
        List<User> users = userDao.queryStudentsByUsernameAndAdmin3(user);
        System.out.println(users);
    }

    @Test
    public void testQueryStudentsByUsernameAndAdmin4(){

        UserDao userDao = session.getMapper(UserDao.class);
        User user=new User();
        //user.setName("武松");
        user.setIsadmin(2);
        List<User> users = userDao.queryStudentsByUsernameAndAdmin4(user);
        System.out.println(users);
    }
    @Test
    public void testUpdateUser(){

        UserDao userDao = session.getMapper(UserDao.class);
        User user=new User();
        user.setId(12L);
        user.setName("李宁");
        user.setPass("111111");
        user.setIsadmin(2);
        userDao.updateUser(user);
```

```
    }

    @Test
    public void testQueryStudentsByIds(){

        UserDao userDao = session.getMapper(UserDao.class);
        List<Long> ids=new ArrayList();
        ids.add(2L);
        ids.add(6L);
        ids.add(10L);
        List<User> users = userDao.queryStudentsByIds(ids);
        System.out.println(users);
    }

    @Test
    public void testInsertBatch(){

        UserDao userDao = session.getMapper(UserDao.class);
        List<User> users=new ArrayList();
        User user1=new User("大乔","123456",1);
        User user2=new User("小乔","123456",0);
        User user3=new User("中乔","123456",1);

        users.add(user1);
        users.add(user2);
        users.add(user3);
        userDao.insertBatch(users);
    }
}
```

SpringMVC笔记：

springmvc是个控制层框架，封装的是servlet

servlet: 接收客户端请求，处理相应的额业务逻辑，最终产生响应给客户端浏览器

mvc： model view controller

模型层 (实体类) 视图层（jsp） 控制层(servlet)

springmvc环境搭建：

1. 创建maven web project

2. pom文件导入依赖

```
<!-- 从这开始 -->
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
  <!-- spring版本号 -->
  <spring.version>4.3.14.RELEASE</spring.version>

  <!-- log4j日志文件管理包版本 -->
  <slf4j.version>1.7.22</slf4j.version>
  <log4j.version>1.2.17</log4j.version>


</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>${log4j.version}</version>
  </dependency>
  <!-- spring -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.13</version>
  </dependency>



  <dependency>
```

```
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-expression</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-instrument</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-instrument-tomcat</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jms</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-messaging</artifactId>
  <version>${spring.version}</version>
</dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-oxm</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
  <version>${spring.version}</version>
</dependency>

<!--springmvc-->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
</dependency>


<!-- mybatis 包 -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.4.6</version>
</dependency>

<!--mybatis spring 插件 -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.3.2</version>
</dependency>

<dependency>
```

```
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.45</version>
</dependency>



<!-- 上传下载 -->
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.2</version>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.2</version>
</dependency>
<dependency>
  <groupId>taglibs</groupId>
  <artifactId>standard</artifactId>
  <version>1.1.2</version>
</dependency>

<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-api</artifactId>
  <version>8.0</version>
  <scope>provided</scope>
</dependency>


<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-web-api</artifactId>
  <version>8.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>net.sf.json-lib</groupId>
  <artifactId>json-lib</artifactId>
  <version>2.1</version>
  <classifier>jdk15</classifier>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>


<!-- pagehelper分页插件 -->
<!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper -->
```

```
<dependency>
  <groupId>com.github.pagehelper</groupId>
  <artifactId>pagehelper</artifactId>
  <version>5.1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.github.jsqlparser/jsqlparser -->
<dependency>
  <groupId>com.github.jsqlparser</groupId>
  <artifactId>jsqlparser</artifactId>
  <version>0.9.5</version>
</dependency>

<!-- poi上传下载组件 -->
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>3.14-beta1</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml-schemas</artifactId>
  <version>3.14-beta1</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>3.14-beta1</version>
</dependency>
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.xmlbeans/xmlbeans -->
<dependency>
  <groupId>org.apache.xmlbeans</groupId>
  <artifactId>xmlbeans</artifactId>
  <version>2.6.0</version>
</dependency>


<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.1.10</version>
</dependency>

<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.30</version>
</dependency>
```

```
</dependencies>

<!-- 到这结束 -->
```

3. web.xml里面配置springmvc相关配置

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>


  <!--部署DispatcherServlet-->
  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <!-- 表示容器在启动时立即加载servlet -->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <!-- 处理所有URL-->
    <url-pattern>/</url-pattern>
  </servlet-mapping>

</web-app>
```

4. 编写Controller方法逻辑

   HelloController

```
package com.hs.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloController {

    @RequestMapping("/hello")
    public String hello(){
        return "hello";
    }
}
```

5. 编写前端页面

   hello.jsp 位于web-inf/jsp目录底下

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

    Hello World!!!

</body>
</html>
```

6. 配置Springmvc配置文件

   springmvc.xml 在resource根目录下创建springmvc.xml文件

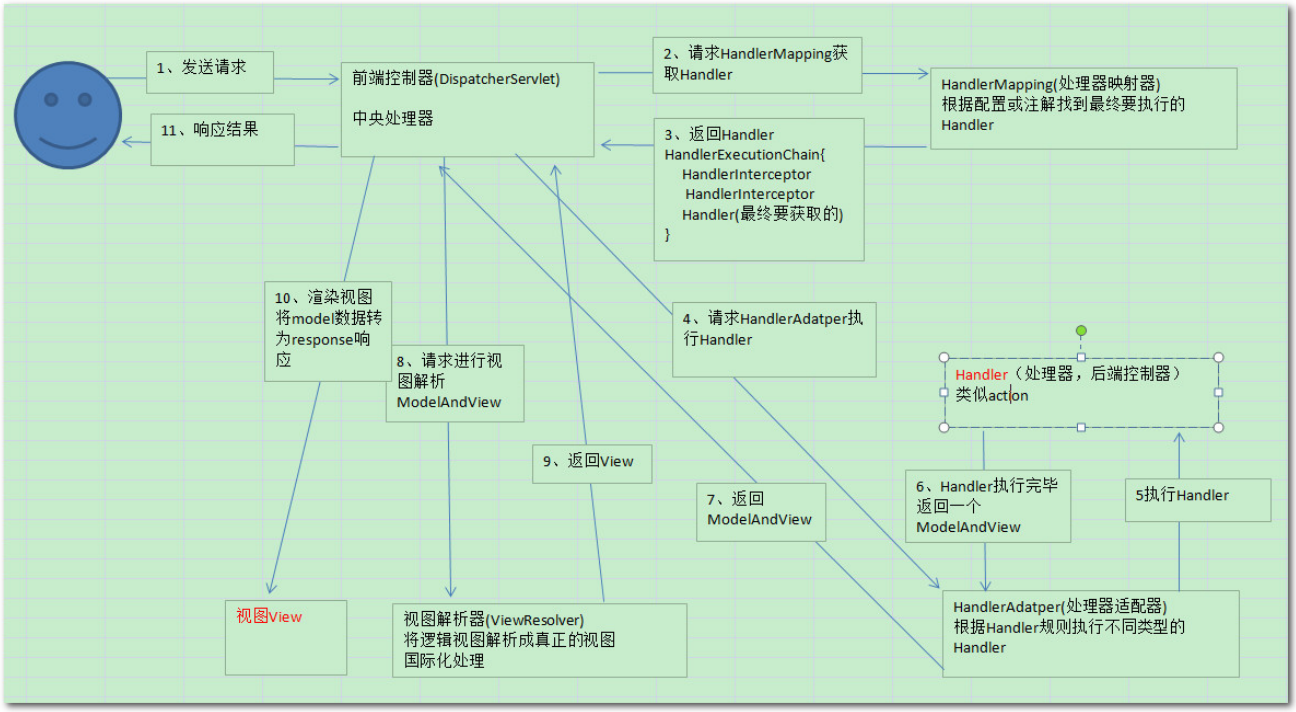
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">


    <!-- 配置自动扫描的包 可以自动识别该包下的所有的注解 -->
    <context:component-scan base-package="com.hs" />


    <!-- 注解驱动 表示对springmvc相关注解的支持 -->
    <mvc:annotation-driven>
```

```
    </mvc:annotation-driven>

    <!-- 配置视图解析器 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

7. 浏览器访问测试，展示效果

   http://localhost:8080/springmvc0100_war/hello

springmvc执行原理（面试题）：



1、 用户发送请求至前端控制器DispatcherServlet。

2、 DispatcherServlet收到请求调用HandlerMapping处理器映射器。

3、 处理器映射器找到具体的处理器(可以根据xml配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

4、 DispatcherServlet调用HandlerAdapter处理器适配器。

5、 HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。

6、 Controller执行完成返回ModelAndView。

7、 HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet。

8、 DispatcherServlet将ModelAndView传给ViewReslover视图解析器。

9、 ViewReslover解析后返回具体View。

10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。

11、 DispatcherServlet响应用户。

1、 http://localhost:8080/springmvc_0100_war/hello

用户再浏览器地址栏敲入url

```
2. 经过DispatcherServlet拦截，进入Springmvc框架

web.xml



  <!--配置springmvc的核心分发器DispatcherServlet-->
  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <!--配置DispatcherServlet的加载时机 1表示tomcat启动的时候加载-->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>


3. 获取request请求的url,和所有的@RequestMapping比对,如果一致，说明我要访问的是他对应的方法controller

由HandlerMapping处理器映射器来完成

找到RequestMapping之后，进入到对应的controller里面，sayHello(),执行逻辑

由HandlerAdapter处理器适配器来完成

HelloController


package com.hs.controller;

import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloController {

    @RequestMapping(value = "/hello")
    public String sayHello(){
        return "hello.jsp";
    }

}
```

4.sayHell()这个controller执行，最后返回值是个视图的字符串 hello.jsp，此时就需要将视图字符串解析成对应的前端视图
通过springmvc.xml里面的解析器来处理

由ViewReslover视图解析器来完成

springmvc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <!--配置组件扫描器-->
    <context:component-scan base-package="com.hs.controller"/>

    <!--配置springmvc的注解驱动-->
    <mvc:annotation-driven/>

    <!--配置视图解析器-->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!--<property name="prefix" value="WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>-->
    </bean>
</beans>
```

5. 最后解析之后再浏览器页面上面相应该视图

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    Hello !!! SpringMVC!!!!
```

```
</body>
</html>
```

springmvc前后端取值传值：

处理post中文乱码问题

在web.xml中配置

```
<!--中文编码过滤器-->
<filter>
  <filter-name>characterEncodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>characterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

取值：

1. login.jsp登录页面

```
<%--
  Created by IntelliJ IDEA.
  User: mumusan
  Date: 2021/4/24
  Time: 16:04
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
```

```
<body>

    <form action="login1" method="post">
        用户名： <input type="text" name="username"><br>
        密码： <input type="password" name="password"><br>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

2. UserController：

```
package com.hs.controller;

import com.hs.pojo.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import javax.servlet.http.HttpServletRequest;

@Controller
public class UserController {

    //通过request对象来取值
    @RequestMapping("/login1")
    public String login1(HttpServletRequest request){
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        System.out.println("用户名： "+username+"密码： "+password);
        return "list";
    }

    //参数名必须和前端表单name属性值保持一致
    @RequestMapping("/login2")
    public String login2(String username,String password){
        System.out.println("用户名： "+username+"密码： "+password);
        return "list";
    }

    //当参数名和属性名不一致时，通过注解@RequestParam来配置对应关系
    @RequestMapping("/login3")
    public String login3(@RequestParam(name = "username") String name, @RequestParam(name =
"password") String pass){
        System.out.println("用户名： "+name+"密码： "+pass);
        return "list";
    }

    //通过javabean来取值，前端表单name属性值必须和实体类属性名称对应一致
    @RequestMapping("/login4")
    public String login4(User user){
        System.out.println("用户名： "+user.getUsername()+"密码： "+user.getPassword());
        return "list";
```

```
    }

}
```

3. 实体类User:

```
package com.hs.pojo;

public class User {
    private Integer id;
    private String username;
    private String password;

        //构造方法，gettersetter方法，toString方法略
}
```

4. 浏览器访问：

http://localhost:8080/springmvc0100_war/login.jsp

传值：

1. login.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

    <form action="login7" method="post">
        用户名： <input type="text" name="username"><br>
        密码： <input type="password" name="password"><br>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

2. UserController

```
//传值
//通过ModelAndView
@RequestMapping("/login5")
public ModelAndView login5(User user){

    ModelAndView modelAndView=new ModelAndView("list");
```

```
    modelAndView.addObject("user",user);
    return modelAndView;
}

//通过Model对象
@RequestMapping("/login6")
public String login6(User user, Model model){

    model.addAttribute("user",user);
    return "list";
}

//通过Map集合
@RequestMapping("/login7")
public String login7(User user, Map map){

    map.put("user",user);
    return "list";
}
```

3. list.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
主页面
用户名：${user.username}
密码：  ${user.password}

</body>
</html>
```

4. 浏览器访问测试

   http://localhost:8080/springmvc0100_war/login.jsp

springmvc类型转换和格式化：

类型转换器：

Spring MVC 框架的 Converter 是一个可以将一种数据类型转换成另一种数据类型的接口，这里 S 表示源类型，T 表示目标类型。也可以自定义类型转换器。

demo: 有一个应用 springMVCDemo 希望用户在页面表单中输入信息来创建商品信息。当输入“apple，10.58，200”时表示在程序中自动创建一个 new Goods，并将“apple”值自动赋给 goodsname 属性，将“10.58”值自动赋给 goodsprice 属性，将“200”值自动赋给 goodsnumber 属性。

1. 前端input.jsp

   input.jsp 注意： 商品数据信息此时以字符串形式某种格式来传输

```
<form action="good" method="post">

    请输入商品信息（格式为apple,10.58,200）
    <input type="text" name="goods"><br/>
    <input type="submit" value="提交">
</form>
```

2. 编写实体类Goods

   Goods:

```
package com.hs.pojo;

public class Goods {
    private String goodsName;
    private double price;
    private int nums;

    public Goods() {
    }

    public Goods(String goodsName, double price, int nums) {
        this.goodsName = goodsName;
        this.price = price;
        this.nums = nums;
    }

    public String getGoodsName() {
        return goodsName;
    }

    public void setGoodsName(String goodsName) {
        this.goodsName = goodsName;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public int getNums() {
        return nums;
```

```
    }

    public void setNums(int nums) {
        this.nums = nums;
    }

    @Override
    public String toString() {
        return "Goods{" +
                "goodsName='" + goodsName + '\'' +
                ", price=" + price +
                ", nums=" + nums +
                '}';
    }
}
```

3. 后台GoodsController取表单值

   注意： 因为前台是所有属性是字符串类型的，后台我直接以Goods对象来接受，是取不到值的，所以需要类型转换，将String转换为Goods，需要自定义类型转换器

```
package com.hs.controller;

import com.hs.pojo.Goods;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class GoodsController {

    @RequestMapping("/good")
    public String myGood(@RequestParam("goods")Goods good, Model model){
        model.addAttribute("good",good);
        return "showGoods";
    }
}
```

4. 前端showGoods.jsp，用于展示controller传过来的数据

   此时是取不到值的，因为类型不一致

```
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

    您输入的商品信息是：${good}
商品名称是： ${good.goodsName}
价格是： ${good.price}
数量： ${good.nums}
</body>
</html>
```

5. 编写自定义类型转换器GoodsConverter

   实现Converter接口 String为源类型，Goods为目标类型

```
package com.hs.convert;

import com.hs.pojo.Goods;
import org.springframework.core.convert.converter.Converter;

public class GoodsConverter implements Converter<String, Goods> {
    @Override
    public Goods convert(String s) {

        //以,拆分
        String[] arrs = s.split(",");
        if(arrs!=null&&arrs.length==3){
            Goods goods=new
Goods(arrs[0],Double.parseDouble(arrs[1]),Integer.parseInt(arrs[2]));
            return goods;
        }
        return null;
    }
}
```

6. 在springmvc配置文件里面配置类型转换器

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">
```

```
    <!-- 配置自动扫描的包 可以自动识别该包下的所有的注解 -->
    <context:component-scan base-package="com.hs" />

    <!--注册类型转换器-->
    <bean id="conversionService"
class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
        <property name="converters">
            <list>
                <bean class="com.hs.convert.GoodsConverter"/>
            </list>
        </property>
    </bean>


    <!-- 注解驱动 表示对springmvc相关注解的支持 -->
    <mvc:annotation-driven conversion-service="conversionService">
    </mvc:annotation-driven>

    <!-- 配置视图解析器 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

7. 浏览器访问，展示运行效果

   http://localhost:8080/springmvc0100_war/input.jsp

格式转换器：

1. 编写addStu.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<form action="addStu" method="post">

    <table>
        <tr>
            <td>学号: </td>
            <td><input type="text" name="stuNo"></td>
        </tr>
        <tr>
```

```
            <td>姓名: </td>
            <td><input type="text" name="stuName"></td>
        </tr>
        <tr>
            <td>生日</td>
            <td><input type="date" name="birthday"></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" name="添加学生"></td>
        </tr>
    </table>
</form>

</body>
</html>
```

2. 编写实体类Student

```
package com.hs.pojo;

import java.util.Date;

public class Student {

    private int stuNo;
    private String stuName;
    private Date birthday;

    public Student() {
    }

    public Student(int stuNo, String stuName, Date birthday) {
        this.stuNo = stuNo;
        this.stuName = stuName;
        this.birthday = birthday;
    }

    public int getStuNo() {
        return stuNo;
    }

    public void setStuNo(int stuNo) {
        this.stuNo = stuNo;
    }

    public String getStuName() {
        return stuName;
    }

    public void setStuName(String stuName) {
        this.stuName = stuName;
    }
```

```
    public Date getBirthday() {
        return birthday;
    }

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

    @Override
    public String toString() {
        return "Student{" +
                "stuNo=" + stuNo +
                ", stuName='" + stuName + '\'' +
                ", birthday=" + birthday +
                '}';
    }
}
```

3. 编写StudentController

   controller里面不能处理date日期类型的数据，会报400错误，类型不匹配，需要格式转换器

```
package com.hs.controller;

import com.hs.pojo.Student;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class StudentController {

    @RequestMapping("/addStu")
    public String addStudent(Student student, Model model){
        System.out.println(student);
        model.addAttribute("stu",student);
        return "showStu";
    }
}
```

4. 编写showStu.jsp，用来显示controller传过来的数据

```
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

学号： ${stu.stuNo}
学生姓名： ${stu.stuName}
```

```
生日: <fmt:formatDate value="${stu.birthday}" pattern="yyyy年MM月dd日"/>

</body>
</html>
```

5. 编写自定义格式转换器DateFormatter

```
package com.hs.convert;

import org.springframework.format.Formatter;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

public class DateFormatter implements Formatter<Date> {

    //日期格式化对象
    private SimpleDateFormat dateFormat;

    public SimpleDateFormat getDateFormat() {
        return dateFormat;
    }

    public void setDateFormat(SimpleDateFormat dateFormat) {
        this.dateFormat = dateFormat;
    }

    //解析
    @Override
    public Date parse(String s, Locale locale) throws ParseException {
        return dateFormat.parse(s);
    }
    //显示
    @Override
    public String print(Date date, Locale locale) {
        return dateFormat.format(date);
    }
}
```

6. springmvc配置格式转换器

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
```

```
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">


    <!-- 配置自动扫描的包 可以自动识别该包下的所有的注解 -->
    <context:component-scan base-package="com.hs" />

    <!--转换器-->
    <bean id="conversionService"
class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
        <!-- 配置自定义类型转换器-->
        <property name="converters">
            <list>
                <bean class="com.hs.convert.GoodsConverter"/>
            </list>
        </property>
        <!-- 配置自定义格式转换器-->
        <property name="formatters">
            <list>
                <bean class="com.hs.convert.DateFormatter">
                    <property name="dateFormat" value="yyyy-MM-dd"/>
                </bean>
            </list>

        </property>
    </bean>


    <!-- 注解驱动 表示对springmvc相关注解的支持 -->
    <mvc:annotation-driven conversion-service="conversionService">
    </mvc:annotation-driven>

    <!-- 配置视图解析器 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

7. 浏览器访问，测试运行效果

   http://localhost:8080/springmvc0100_war/addStu.jsp

springmvc统一异常的处理：

统一处理某一类异常，能够减少代码的重复度和复杂度，有利于代码的维护。

1. 局部控制

@ExceptionHandler

使用@ExceptionHandler注解作用在方法上面，参数是具体的异常类型。一旦系统抛出这种类型的异常时，会引导到该方法来处理。但是它的缺陷很明显

处理异常的方法和出错的方法(或者异常最终抛出来的地方)必须在同一个controller，不能全局控制。

FFFController：

```
package com.hs.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class FFFController {


    @RequestMapping("error002")
    @ResponseBody
    public String error002(){
        int i=1/0;
        return null;
    }


}
```

EEEController：

```
package com.hs.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class EEEController {


    @RequestMapping("error001")
    @ResponseBody
    public String error001(){
        int i=1/0;
        return null;
    }
```

```
    @ExceptionHandler(RuntimeException.class)
    @ResponseBody
    public String error003(){
        return "runtime exception...";
    }
}
```

测试效果：

当访问error001时，异常能够处理；当访问error002时，异常不能处理，所以处理异常的方法和出错的方法(或者异常最终抛出来的地方)必须在同一个controller，不能全局控制

2. 全局处理

@ControllerAdvice @ExceptionHandler

使用@ControllerAdvice 和@ExceptionHandler 可以全局控制异常，使业务逻辑和异常处理分隔开。

GlobalExceptionHandler：

```
package com.hs.controller;

import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(Exception.class)
    public String handlerAllException(Exception e){
        System.out.println("handler Exception");
        return "error";
    }
}
```

error.jsp：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
处理所有的异常信息
</body>
</html>
```

此时再访问error002,异常就能处理，会自动跳转到error.jsp异常页面

SSM环境搭建：

1. 概述：

在写代码之前我们先了解一下这三个框架分别是干什么的?

1. SpringMVC：它用于web层，相当于controller（等价于传统的servlet和struts的action），用来处理用户请求。举个例子，用户在地址栏输入http://网站域名/login，那么springmvc就会拦截到这个请求，并且调用controller层中相应的方法，（中间可能包含验证用户名和密码的业务逻辑，以及查询数据库操作，但这些都不是springmvc的职责），最终把结果返回给用户，并且返回相应的页面（当然也可以只返回json/xml等格式数据）。springmvc就是做前面和后面过程的活，与用户打交道！！
2. Spring：太强大了，以至于我无法用一个词或一句话来概括它。但与我们平时开发接触最多的估计就是IOC容器，它可以装载bean（也就是我们java中的类，当然也包括service dao里面的），有了这个机制，我们就不用在每次使用这个类的时候为它初始化，很少看到关键字new。另外spring的aop，事务管理等等都是我们经常用到的。
3. MyBatis：数据访问层框架，封装的是jdbc; 如果你问我它跟鼎鼎大名的Hibernate有什么区别？我只想说，他更符合我的需求。第一，它能自由控制sql，这会让有数据库经验的人（当然不是说我啦~捂脸~）编写的代码能搞提升数据库访问的效率。第二，它可以使用xml的方式来组织管理我们的sql，因为一般程序出错很多情况下是sql出错，别人接手代码后能快速找到出错地方，甚至可以优化原来写的sql。

2. SSM框架整合配置

2.1 开发环境

IDE: idea 2020.1

JDK: 1.8

MySql: 5.7

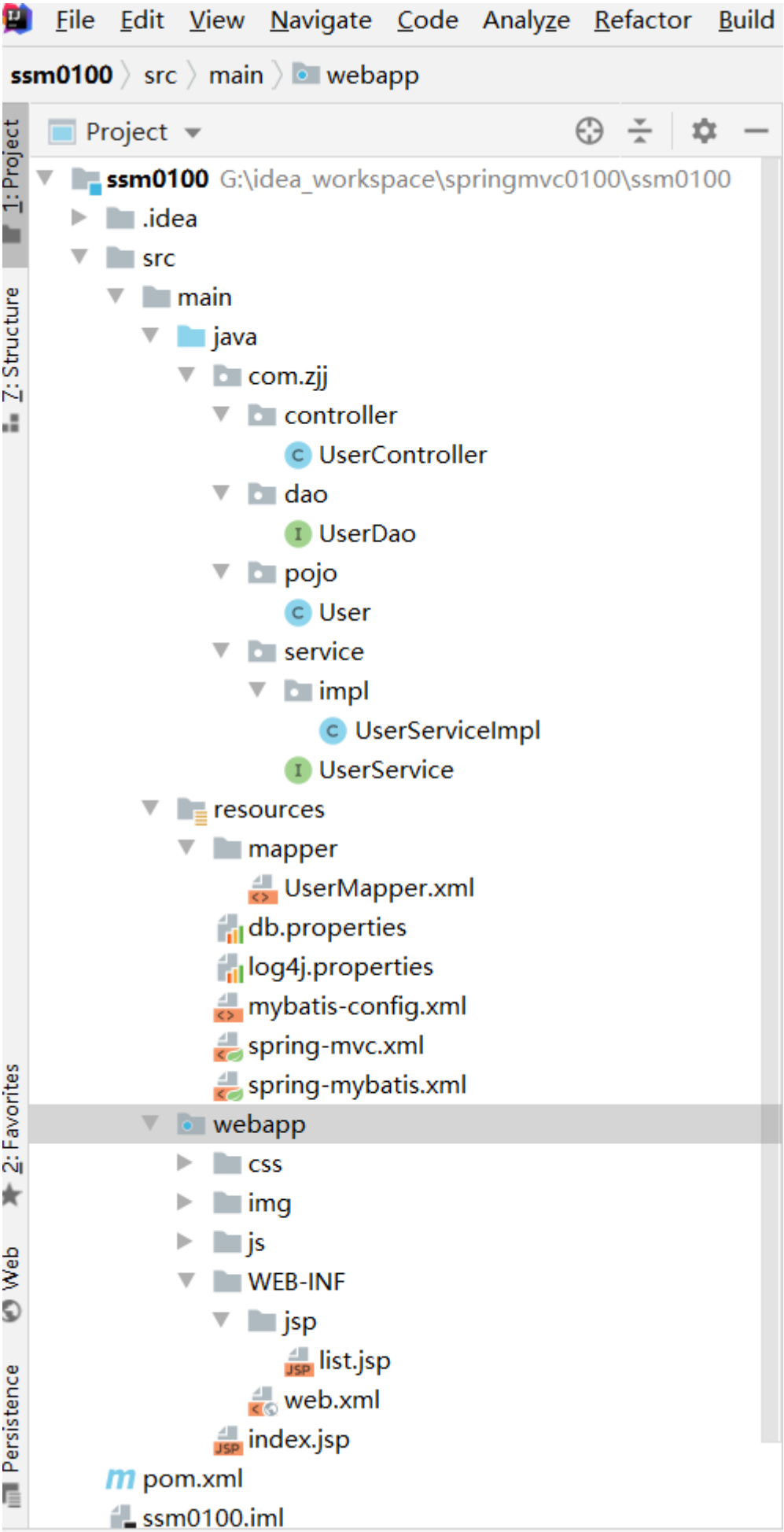
tomcat: 9版本

Maven： 3.6.0

2.2 完整的项目结构

配置文件，参考demo已经上传U+平台，注意参考

完成的项目结构是这样子的


File Edit View Navigate Code Analyze Refactor Build
ssm0100 src main webapp
Project
1: Project
7: Structure
2: Favorites
Web
Persistence
ssm0100 G:\idea_workspace\springmvc0100\ssm0100
.idea
src
main
java
com.zjj
controller
UserController
dao
UserDao
pojo
User
service
impl
UserServiceImpl
UserService
resources
mapper
UserMapper.xml
db.properties
log4j.properties
mybatis-config.xml
spring-mvc.xml
spring-mybatis.xml
webapp
css
img
js
WEB-INF
jsp
list.jsp
web.xml
index.jsp
pom.xml
ssm0100.iml

配置文件说明：

log4j.properties： 日志配置文件

db.properties： 数据库连接的数据源配置信息

mybatis-config.xml： mybatis配置文件

UserMapper.xml： mybatis映射文件

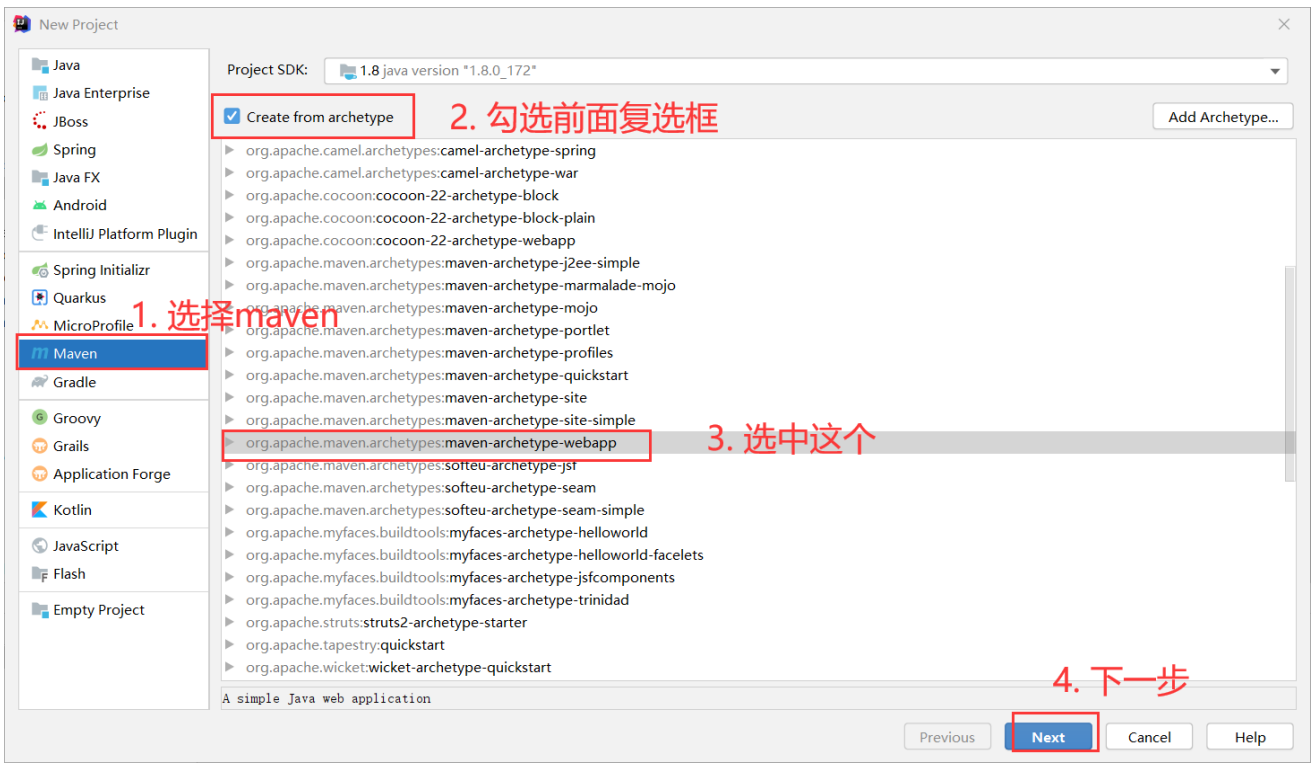
spring-mybatis.xml: spring和mybatis整合的配置文件，将mybatis的相关配置通过spring配置的方式来实现注入

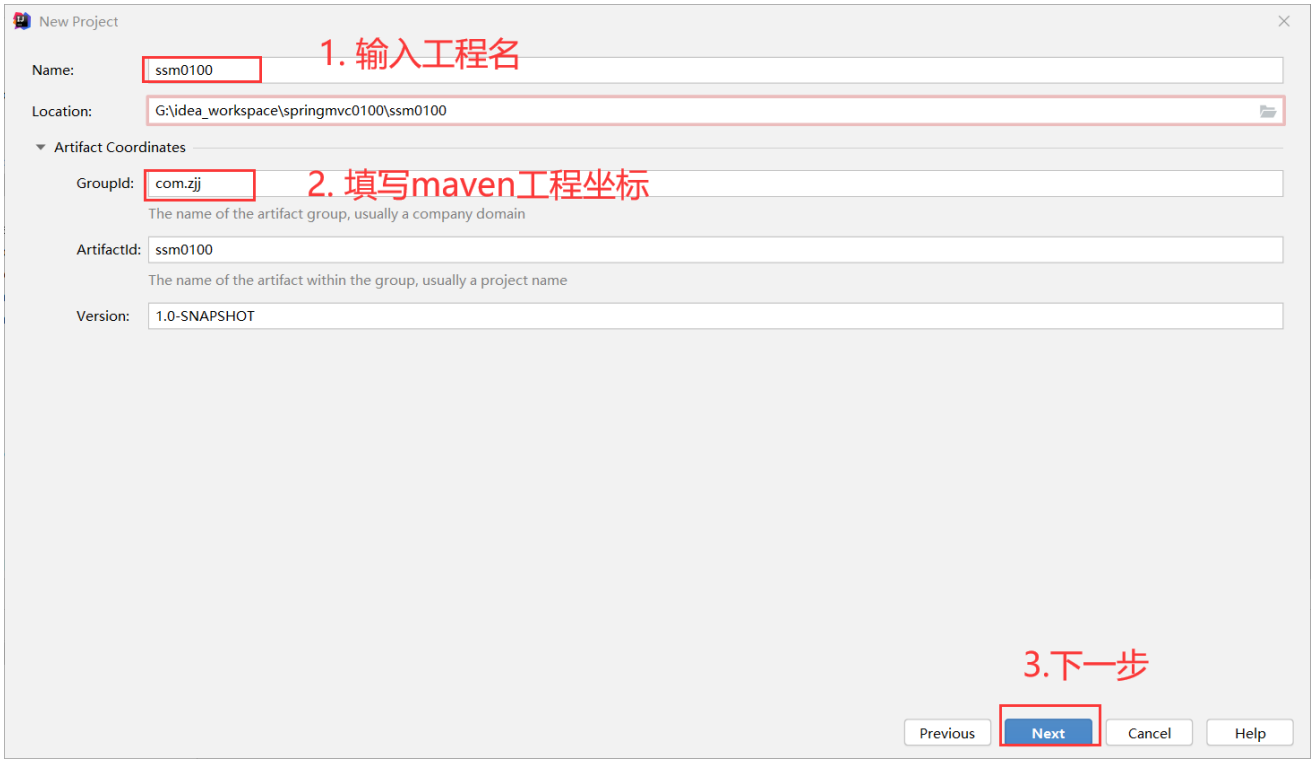
spring-mvc.xml： springmvc的配置文件

web.xml web工程的核心配置文件

3. 创建Maven WEB工程

File---->new project------>Maven----->勾选 Create from archetype复选框------>选择 maven-archetype-webapp--next--->输入 project name ,maven坐标---->next------->finish





下一步，finish完成即可

注意： 前提是idea已经关联好本机maven，使用maven时一定要联网，一定要等项目创建成功

创建成功之后，进入到 工程的pom.xml文件里面，吧下面相关内容删掉，从name标签到bulid标签删掉

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.hs</groupId>
  <artifactId>ssm0100</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <!--从这开始 -->
  <!--开始结束中间内容删掉，这些是自动生成的，用不着 -->
  <name>ssm0100 Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>
```

```
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>ssm01</finalName>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be
moved to parent pom) -->
      <plugins>
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.1.0</version>
        </plugin>
        <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.22.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>3.2.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.5.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.8.2</version>
        </plugin>
      </plugins>
    </pluginManagement>
```

```
  </build>

  <!--到这结束 -->
</project>
```

下一步： 吧maven工程的src/main/java; src/main/resources; src/test/java目录补全，没有的话新建即可

4. 在pom.xml文件中导入所需的所有的依赖

从这开始---到这结束 中间内容 复制进去尽可以

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.hs</groupId>
  <artifactId>ssm0100</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <!-- 从这开始 -->

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
    <!-- spring版本号 -->
    <spring.version>4.3.14.RELEASE</spring.version>
    <!-- log4j日志文件管理包版本 -->
    <slf4j.version>1.7.22</slf4j.version>
    <log4j.version>1.2.17</log4j.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
```

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>${log4j.version}</version>
</dependency>
<!-- spring -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
  <version>${spring.version}</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.8.13</version>
</dependency>



<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-expression</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
```

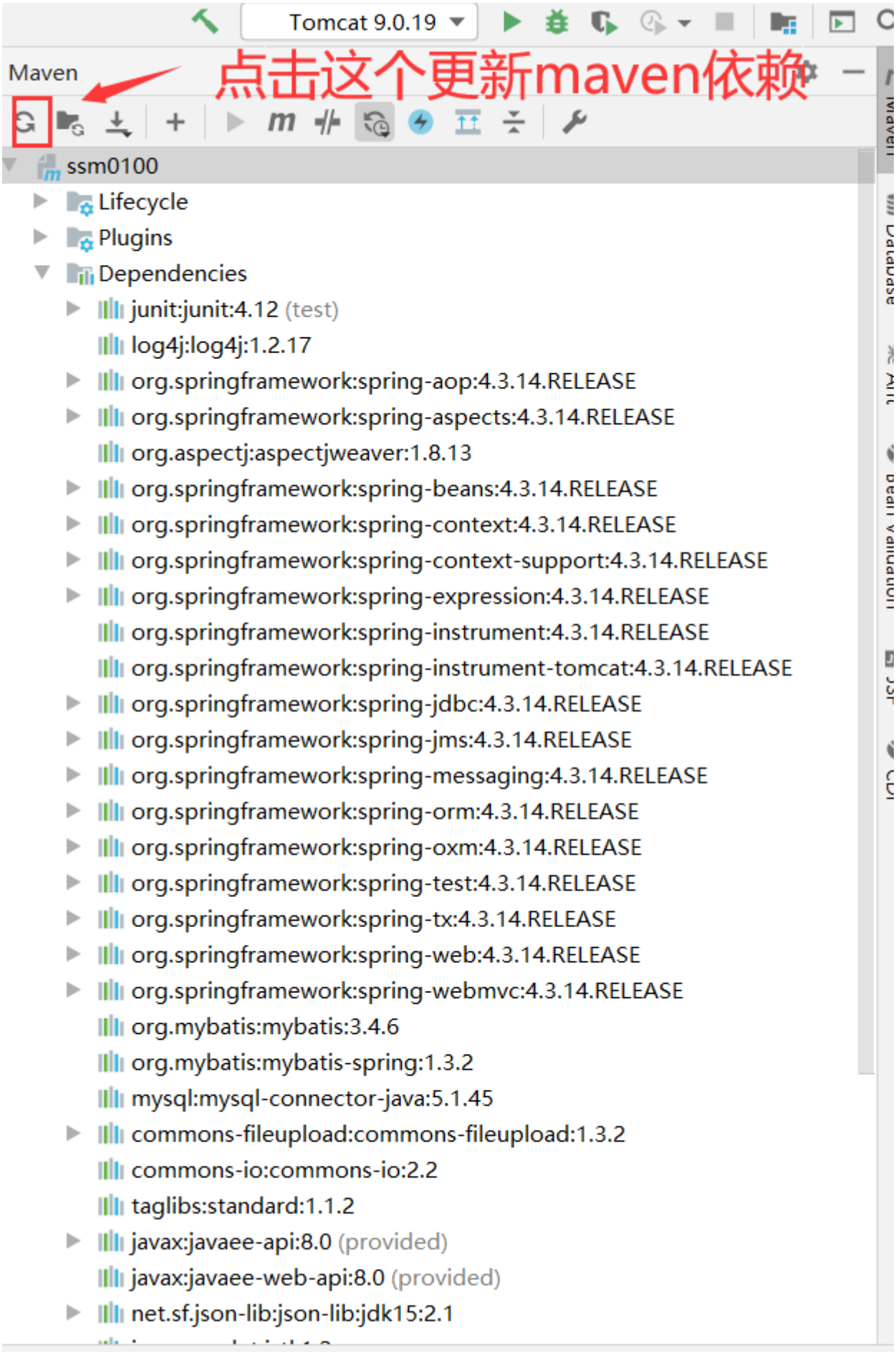
```
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument-tomcat</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-messaging</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-oxm</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring.version}</version>
  </dependency>
```

```
<!-- springmvc -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
</dependency>


<!-- mybatis 包 -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.4.6</version>
</dependency>

<!--mybatis spring 插件 -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.3.2</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.45</version>
</dependency>

<!-- 上传下载 -->
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.2</version>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.2</version>
</dependency>
<dependency>
  <groupId>taglibs</groupId>
  <artifactId>standard</artifactId>
  <version>1.1.2</version>
</dependency>

<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-api</artifactId>
```

```
    <version>8.0</version>
    <scope>provided</scope>
  </dependency>


  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-web-api</artifactId>
    <version>8.0</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>net.sf.json-lib</groupId>
    <artifactId>json-lib</artifactId>
    <version>2.1</version>
    <classifier>jdk15</classifier>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>


  <!-- pagehelper分页插件 -->
  <!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper -->
  <dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>5.1.2</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/com.github.jsqlparser/jsqlparser -->
  <dependency>
    <groupId>com.github.jsqlparser</groupId>
    <artifactId>jsqlparser</artifactId>
    <version>0.9.5</version>
  </dependency>

  <!-- poi上传下载组件 -->
  <dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>3.14-beta1</version>
  </dependency>
  <dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml-schemas</artifactId>
    <version>3.14-beta1</version>
  </dependency>
  <dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.14-beta1</version>
```

```
    </dependency>
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>4.5.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.apache.xmlbeans/xmlbeans -->
    <dependency>
      <groupId>org.apache.xmlbeans</groupId>
      <artifactId>xmlbeans</artifactId>
      <version>2.6.0</version>
    </dependency>

    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>druid</artifactId>
      <version>1.1.10</version>
    </dependency>

    <!--fastjson-->
    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>fastjson</artifactId>
      <version>1.2.30</version>
    </dependency>


  </dependencies>


  <build>
    <pluginManagement>
      <plugins>
        <!-- 配置Tomcat插件 -->
        <plugin>
          <groupId>org.apache.tomcat.maven</groupId>
          <artifactId>tomcat7-maven-plugin</artifactId>
          <version>2.2</version>
          <configuration>
            <port>8082</port>
            <path>/</path>
          </configuration>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>


  <!-- 到这结束 -->


</project>
```

导入之后记得更新maven依赖


Tomcat 9.0.19
Maven
点击这个更新maven依赖
ssm0100
Lifecycle
Plugins
Dependencies
junit:junit:4.12 (test)
log4j:log4j:1.2.17
org.springframework:spring-aop:4.3.14.RELEASE
org.springframework:spring-aspects:4.3.14.RELEASE
org.aspectj:aspectjweaver:1.8.13
org.springframework:spring-beans:4.3.14.RELEASE
org.springframework:spring-context:4.3.14.RELEASE
org.springframework:spring-context-support:4.3.14.RELEASE
org.springframework:spring-expression:4.3.14.RELEASE
org.springframework:spring-instrument:4.3.14.RELEASE
org.springframework:spring-instrument-tomcat:4.3.14.RELEASE
org.springframework:spring-jdbc:4.3.14.RELEASE
org.springframework:spring-jms:4.3.14.RELEASE
org.springframework:spring-messaging:4.3.14.RELEASE
org.springframework:spring-orm:4.3.14.RELEASE
org.springframework:spring-oxm:4.3.14.RELEASE
org.springframework:spring-test:4.3.14.RELEASE
org.springframework:spring-tx:4.3.14.RELEASE
org.springframework:spring-web:4.3.14.RELEASE
org.springframework:spring-webmvc:4.3.14.RELEASE
org.mybatis:mybatis:3.4.6
org.mybatis:mybatis-spring:1.3.2
mysql:mysql-connector-java:5.1.45
commons-fileupload:commons-fileupload:1.3.2
commons-io:commons-io:2.2
taglibs:standard:1.1.2
javax:javaee-api:8.0 (provided)
javax:javaee-web-api:8.0 (provided)
net.sf.json-lib:json-lib:jdk15:2.1

5. 整合思路

1、Dao层：

mybatis-config.xml： Mybatis的配置文件；可以配置mybatis的日志信息，别名配置，分页插件等；

文件必须存在。

spring-mybatis.xml：mybatis整合spring，通过由spring创建数据库连接池，spring管理SqlSessionFactory、mapper代理对象。需要mybatis和spring的整合包。

2、Service层：

3、控制层： Springmvc框架，由springmvc管理controller

spring-mvc.xml：springmvc配置文件

6. 加入配置文件

配置文件说明：

log4j.properties： 日志配置文件

db.properties： 数据库连接的数据源配置信息

mybatis-config.xml： mybatis配置文件

UserMapper.xml： mybatis映射文件

spring-mybatis.xml: spring和mybatis整合的配置文件，将mybatis的相关配置通过spring配置的方式来实现注入

spring-mvc.xml： springmvc的配置文件

web.xml web工程的核心配置文件

记得所有配置搭完之后最后再启动服务器；启动服务器 不能有报错信息。如果有报错提示，说明配置信息有问题

1. db.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/b?characterEncoding=utf-8
jdbc.username=root
jdbc.password=123456
```

注意： 其中url中数据库名，数据库密码一定要和自己的保持一致

2. log4j.properties

里面内容无需改动，直接复制即可；配置日志信息，执行操作时可以看到sql语句及参数

```
log4j.rootLogger=debug,stdout,logfile
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout
log4j.logger.com.ibatis=DEBUG
log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG
log4j.logger.com.ibatis.common.jdbc.ScriptRunner=DEBUG
log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=DEBUG
log4j.logger.Java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
```

3. mybatis-config.xml

mybatis配置文件；注意不要忘了setting里面配置mybatis-3-config.dtd 这个，否则标签没有提示功能

注意： 别名的配置包名一定要和自己的pojo实体类包名保持一致；

pagehelper分页插件是mybatis的一个分页插件，可以很好地实现分页功能, 配上吧

```
<?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>


    <!-- 配置日志信息 -->
    <settings>
        <setting name="logImpl" value="STDOUT_LOGGING"/>
    </settings>

   <!-- 别名配置方式二 -->
   <typeAliases>
      <!-- 配置该包底下的所有的类默认使用类名表示别名 -->
      <package name="com.hs.pojo"/>
   </typeAliases>

   <!-- 配置分页插件 -->
    <plugins>
        <plugin interceptor="com.github.pagehelper.PageInterceptor">
            <!-- 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库-->
            <property name="helperDialect" value="mysql"/>
            <property name="reasonable" value="true"/>
        </plugin>
    </plugins>

</configuration>
```

4. spring-mybatis.xml

spring和mybatis整合的配置文件，其实就是把mybatis的sqlSessionFactory,数据源，事务等信息通过spring来给他注入

注意： 这个配置 包名一定要和自己dao的包名保持一致

映射文件默认在resources/mapper/*Mapper.xml 注意路径和文件名写法；重点是**配置跟实际路径保持一致**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.3.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">


    <!-- 引入外部properties文件 -->
    <context:property-placeholder location="classpath:/db.properties" />

    <!-- 配置数据源 -->
    <bean class="org.springframework.jdbc.datasource.DriverManagerDataSource"
id="dataSource">
        <property name="driverClassName" value="${jdbc.driver}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <!-- 配置SqlSessionFactory对象 -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <!-- 引入mybatis配置文件 -->
        <property name="configLocation" value="classpath:mybatis-config.xml" />
        <!-- 引入mybatis映射文件 -->
        <property name="mapperLocations" value="classpath:mapper/*Mapper.xml"/>
    </bean>

    <!-- 自动扫描 将Mapper接口生成代理注入到Spring 基于xml方式的-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
```

```
        <property name="basePackage" value="com.hs.dao" />
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
    </bean>

    <!-- 配置引入映射文件 基于注解方式的 -->
    <!--
    <bean class="org.mybatis.spring.mapper.MapperFactoryBean">
        <property name="mapperInterface" value="com.hs.dao.UserDao" />
        <property name="sqlSessionFactory" ref="sqlSessionFactory" />
    </bean>
     -->

    <!-- 配置事务管理器 -->
    <bean id="transactionManager"
          class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 配置事务的注解驱动  -->
    <tx:annotation-driven/>

</beans>
```

5. web.xml

在WEB-INF目录下，这个文件一定要配置，不能忘记；**注意注意注意！！！**

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>


  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-*.xml</param-value>
  </context-param>
  <!-- 配置中文乱码过滤器 -->
  <filter>

    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
```

```
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>characterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- 配置spring在web项目中的使用 配置spring的监听器 -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- 配置springmvc的核心分发器 -->
<servlet>
  <servlet-name>springmvc</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <!-- 引入springmvc的配置文件 -->
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-mvc.xml</param-value>
  </init-param>
  <!-- 配置servlet的加载时机 1表示tomcat启动的时候加载 -->
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>springmvc</servlet-name>
  <!-- / 代表项目名底下所有 -->
  <url-pattern>/</url-pattern>
</servlet-mapping>

<!-- ssm配置静态资源访问2 某种类型的资源文件 的访问-->
<servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.jpg</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.gif</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.css</url-pattern>
</servlet-mapping>
```

```
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.js</url-pattern>
  </servlet-mapping>

</web-app>
```

6. spring-mvc.xml

springmvc的配置文件

注意： 包名一定写成所有组件的父包

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">


    <!-- 配置自动扫描的包 可以自动识别该包下的所有的注解 -->
    <context:component-scan base-package="com.hs" />

    <!-- 注解驱动 表示对springmvc相关注解的支持 -->
    <!-- json 转换器 -->
    <mvc:annotation-driven>
        <mvc:message-converters>
            <bean

class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter">
                <description>JSON转换器</description>
                <property name="supportedMediaTypes">
                    <list>
                        <value>application/json;charset=UTF-8</value>
                        <value>text/html;charset=UTF-8</value>
                    </list>
                </property>
            </bean>
        </mvc:message-converters>
    </mvc:annotation-driven>

    <!-- 配置视图解析器 -->
    <bean
            class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
```

```
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- 配置类似于全局的拦截器  如果项目中用到拦截器，需要配此项，如果没用的到拦截器，不能配，否则会报错--
>
    <!--
    <mvc:interceptors>
        <mvc:interceptor>

            <mvc:mapping path="/**"/>

            <mvc:exclude-mapping path="/login"/>
            <bean class="com.bw.util.PermissionIntercepter"></bean>
       </mvc:interceptor>
    </mvc:interceptors>
 -->
    <!-- 配置上传解析器 -->
    <!-- 设置文件上传大小 -->
    <!-- 配置编码 -->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <property name="MaxUploadSize">
            <value>6000000</value>
        </property>
        <property name="defaultEncoding" value="UTF-8"/>
    </bean>

    <!--静态资源的处理-->
   <mvc:default-servlet-handler/>

    <!-- 静态资源访问配置
    1. 在springmvc配置文件里面配某个目录的访问
    2. 在web.xml文件里面配置某种后缀名的文件可以访问
    两种配置方式选择一种
    -->

    <!-- 配置静态资源的映射 -->
   <!-- <mvc:resources location="/css/" mapping="/css/**" />
    <mvc:resources location="/js/" mapping="/js/**" />
    <mvc:resources location="/img/" mapping="/img/**" />
    -->

</beans>
```

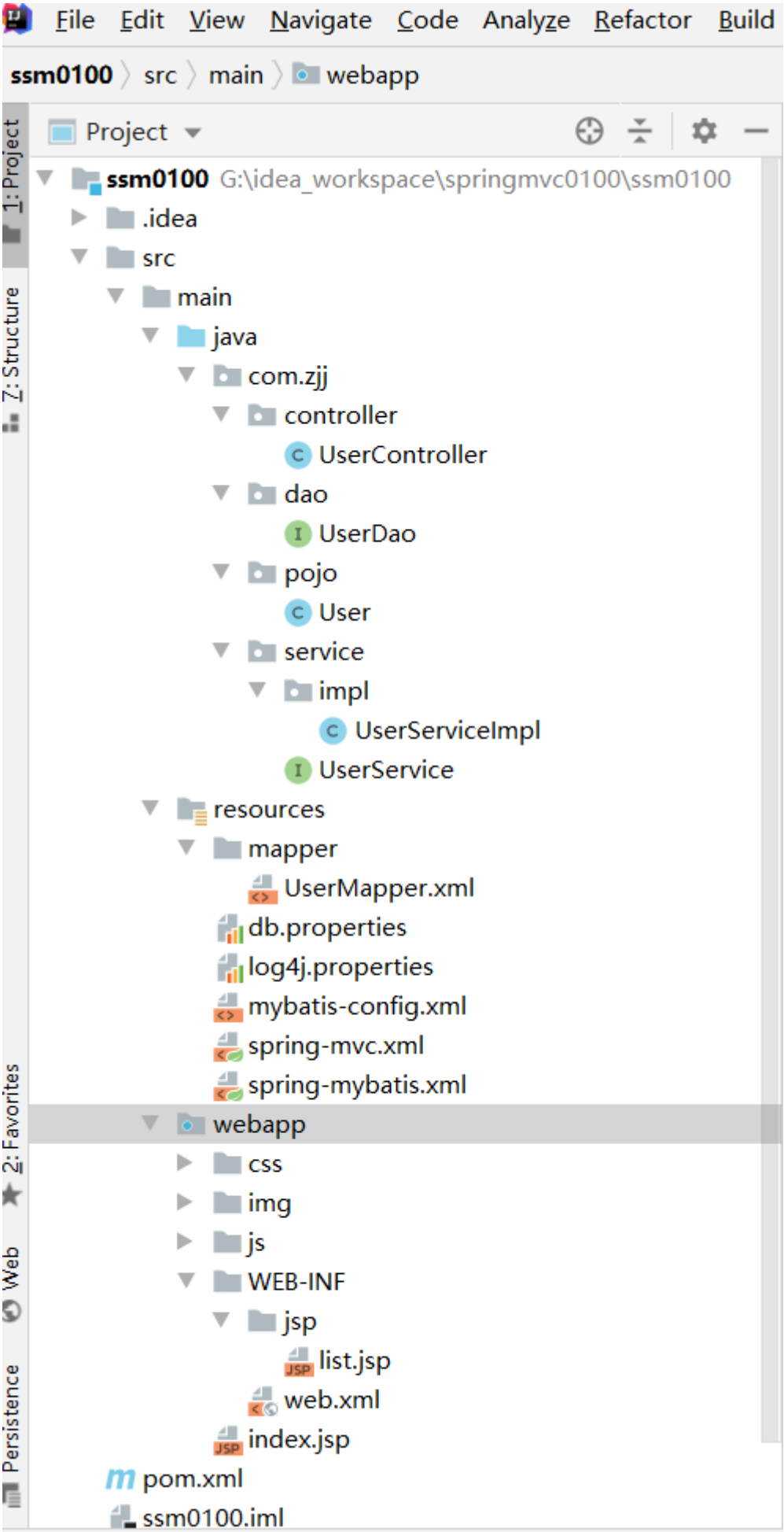
7. 环境搭建

到这里，其实整个框架的搭建已经基本了，毕竟整点就是上边这堆配置文件嘛，接下来，编写每层的代码。

参考如下架构

父包：

com.hs 下面子包 pojo,dao,service,controller


File Edit View Navigate Code Analyze Refactor Build
ssm0100 src main webapp
Project
1: Project
7: Structure
2: Favorites
Web
Persistence
ssm0100 G:\idea_workspace\springmvc0100\ssm0100
.idea
src
main
java
com.zjj
controller
UserController
dao
UserDao
pojo
User
service
impl
UserServiceImpl
UserService
resources
mapper
UserMapper.xml
db.properties
log4j.properties
mybatis-config.xml
spring-mvc.xml
spring-mybatis.xml
webapp
css
img
js
WEB-INF
jsp
list.jsp
web.xml
index.jsp
pom.xml
ssm0100.iml

6: TODO　8: Services　Terminal　Java Enterprise

1. 数据库表

对象　stu @test (mysql01) - 表　tb_user @b (mysql01) - 表　tb_user @b (mysql01) - 表

新建　保存　另存为　添加字段　插入字段　删除字段　主键　上移　下移

字段　索引　外键　触发器　选项　注释　SQL 预览

名	类型	长度	小数点	不是 null	
id	bigint	20	0	☑	🔑1
username	varchar	255	0	☐	
password	varchar	255	0	☐	
isadmin	int	11	0	☐	

默认:

注释: 用户编号 ...

☑ 自动递增　主键自增

☐ 无符号

☐ 填充零

对象 stu @test (mysql01) - 表 tb_user @b (mysql01) - 表

开始事务 备注 筛选 排序 导入 导出

id	username	password	isadmin
1	貂蝉	123456	1
2	李四2	111211	2
3	wangwu	111111	0
5	武松	7843784	1
6	武松	7843784	2
7	关羽	123456	1
10	吕布	123456	1
12	李宁	111111	2
14	aaa	111	1
15	bbb	222	1
16	大乔	123456	1
17	小乔	123456	0
18	中乔	123456	1
19	吕布2	123456	1
20	董卓	111111	1
21	关羽1	123456	1
22	关羽2	123456	1
23	公孙瓒	123456	1
24	公孙离	123456	1
25	公孙止	654321	1

2. pojo实体类User

User:

```
package com.hs.pojo;

public class User {


    private Long id;
    private String username;
    private String password;
    private Integer isadmin;

    public User() {
    }

    public User(Long id, String username, String password, Integer isadmin) {
        this.id = id;
```

```
        this.username = username;
        this.password = password;
        this.isadmin = isadmin;
    }

    public User(String username, String password, Integer isadmin) {
        this.username = username;
        this.password = password;
        this.isadmin = isadmin;
    }

    public Long getId() {
        return id;
    }

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

    public String getUsername() {
        return username;
    }

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

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getIsadmin() {
        return isadmin;
    }

    public void setIsadmin(Integer isadmin) {
        this.isadmin = isadmin;
    }

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

3. dao层接口

UserDao:

```
package com.hs.dao;

import com.hs.pojo.User;

import java.util.List;



public interface UserDao {
    //查询所有
    public List<User> queryAllUsers();

}
```

4. mybatis映射文件

UserMapper.xml 注意路径 位于 resources/mapper目录下

注意： namespace 值必须和接口路径保持一致

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace 值必须和接口路径保持一致-->
<mapper namespace="com.hs.dao.UserDao">
    <!--查询所有 -->
    <select id="queryAllUsers" resultType="User">
        select * from tb_user
    </select>
</mapper>
```

5. service层代码

UserService：

```
package com.hs.service;

import com.hs.pojo.User;

import java.util.List;

public interface UserService {

    //查询所有
    public List<User> queryAllUsers();

}
```

UserServiceImpl：

注意：@Service @Transactional 注解的配置

```
package com.hs.service.impl;

import com.hs.dao.UserDao;
import com.hs.pojo.User;
import com.hs.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
@Transactional
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;

    @Override
    public List<User> queryAllUsers() {
        return userDao.queryAllUsers();
    }

}
```

6. controller层代码

UserController：

```
package com.hs.controller;
```

```
import com.hs.pojo.User;
import com.hs.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.List;

@Controller
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserService userService;

    @RequestMapping("/list")
    public String queryAllUsers(Model model){
        List<User> users = userService.queryAllUsers();
        model.addAttribute("list",users);
        return "list";
    }
}
```

7. 前端页面

list.jsp页面位于 WEB-INF/jsp目录底下，视图解析器里面有相应的前缀和后缀的配置

```
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

<center>
    <h1>学生信息管理系统</h1>
    <table width="60%" height="40%">
        <tr>
            <td>编号</td>
            <td>姓名</td>
            <td>密码</td>
            <td>管理员</td>
        </tr>
        <c:forEach items="${list}" var="u">
            <tr>
                <td>${u.id}</td>
                <td>${u.username}</td>
                <td>${u.password}</td>
                <td>${u.isadmin}</td>
```

```
            </tr>
        </c:forEach>
    </table>
</center>
</body>
</html>
```

8. 启动tomcat服务器，浏览器访问，测试运行效果

url: http://localhost:8080/ssm0100_war/user/list

运行效果：

编号	姓名	密码	管理员
1	貂蝉	123456	1
2	李四2	111211	2
3	wangwu	111111	0
5	武松	7843784	1
6	武松	7843784	2
7	关羽	123456	1
10	吕布	123456	1
12	李宁	111111	2
14	aaa	111	1
15	bbb	222	1
16	大乔	123456	1
17	小乔	123456	0
18	中乔	123456	1
19	吕布2	123456	1
20	董卓	111111	1
21	关羽1	123456	1
22	关羽2	123456	1
23	公孙瓒	123456	1
24	公孙离	123456	1
25	公孙止	654321	1

环境搭建成功！！！

后续功能点：

登录 注册 增删改查 模糊查询 区间查询 分页（pagehelper分页插件）

demo1: 使用SSM实现用的的登录和注册功能：

springmvc里面正常返回值后面会加视图解析器的前缀和后缀；

但是请求转发和重定向不会加前缀和后缀，直接跳转到forward或者redirect的地址

springmvc实现请求转发和重定向的方式：

return "forward:地址" 请求转发

return "redirect:地址" 重定向

ssm环境搭建

大前提，环境搭建搭建成功，直接往里面填写代码

eg1：实现登录功能

1. 数据库表user

   还使用环境搭建的tb_user表 id username password isadmin

2. 编写pojo实体类 User

```
public class User {


    private Long id;
    private String username;
    private String password;
    private Integer isadmin;
    //................
}
```

3. login.jsp 放在web根路径下，它是入口文件

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<center>
    <h1>登录页面</h1>
    <form action="login" method="post">
    <table>
        <tr>
            <td>用户名</td>
            <td><input type="text" name="username"></td>
        </tr>
        <tr>
            <td>密码</td>
```

```
            <td><input type="password" name="password"></td>
        </tr>
        <tr>
            <td><input type="submit" value="登录"></td>
            <td><a href="register.jsp">立即注册</a></td>
        </tr>
    </table>
    </form>
</center>

</body>
</html>
```

4. dao层接口及映射文件配置

   UserDao

```
public User queryUserByUsernameAndPassword(User user);
```

   UserServiceImpl

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace 值必须和接口路径保持一致-->
<mapper namespace="com.hs.dao.UserDao">

    <resultMap id="userResultMap" type="User">

    </resultMap>
    <!-- 登录 -->
    <select id="queryUserByUsernameAndPassword" resultMap="userResultMap"
parameterType="User">
        select * from tb_user where username=#{username} and password=#{password}
    </select>

</mapper>
```

5. service层接口及其实现

   UserService：

```
package com.hs.service;

import com.hs.pojo.User;

public interface UserService {


    //登录
    public User queryUserByUsernameAndPassword(User user);

}
```

UserServiceImpl：

```
package com.hs.service.impl;

import com.hs.dao.UserDao;
import com.hs.pojo.User;
import com.hs.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;


    @Override
    public User queryUserByUsernameAndPassword(User user) {
        return userDao.queryUserByUsernameAndPassword(user);
    }


}
```

6. controller中login()方法

UserController：

```
package com.hs.controller;

import com.hs.pojo.User;
import com.hs.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class UserController {

    @Autowired
    private UserService userService;

    @RequestMapping("/login")
    public String login(User user,Model model){
        User htUser=userService.queryUserByUsernameAndPassword(user);

        return "stulist";
    }
```

```
}
```

7. 主页面stulist.jsp： 登录成功跳转主页面 位于web/inf/jsp目录底下

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
主页面
</body>
</html>
```

8. 浏览器访问测试

   http://localhost:8080/ssm0100_war/login.jsp

eg2：实现注册功能

1. register.jsp 位于web根目录底下

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<center>
    <h1>注册页面</h1>
    <form action="register" method="post">
    <table>
        <tr>
            <td>用户名</td>
            <td><input type="text" name="username"></td>
        </tr>
        <tr>
            <td>密码</td>
            <td><input type="password" name="password"></td>
        </tr>
        <tr>
            <td><input type="radio" name="isadmin" value="1">管理员</td>
            <td><input type="radio" name="isadmin" value="2" checked>普通用户</td>
        </tr>
        <tr>
            <td><input type="submit" value="注册"></td>
            <td><input type="reset" value="重置"></td>
        </tr>
    </table>
    </form>
</center>
```

```
</body>
</html>
```

2. dao层接口及映射文件配置

   UserDao：

```
public int insertUser(User user);
```

   UserMapper.xml：

```
<!--注册-->
<insert id="insertUser" parameterType="User">
    insert into tb_user values (0,#{username},#{password},#{isadmin})
</insert>
```

3. service层接口及其实现

   UserService：

```
//注册
public boolean addUser(User user);
```

   UserServiceImpl：

```
@Override
public boolean addUser(User user) {
    int row = userDao.insertUser(user);
    if(row>0){
        return true;
    }
    return false;
}
```

4. controller中register()方法

   注册成功，重定向到login.jsp； 注册失败，回到register.jsp

```
@RequestMapping("/register")
public String register(User user,Model model){
    boolean flag=userService.addUser(user);
    if(flag){
        return "redirect:login.jsp";
    }
    return "redirect:register.jsp";
}
```

5. 浏览器访问测试

   http://localhost:8080/ssm0100_war/register.jsp

demo2: 实现查询所有学生信息功能

1. 设计数据表表tb_stu

   int sid; varchar sname; date birthday;varchar address ; varchar photo

2. 编写实体类Student

```
package com.hs.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {
    private Integer id;
    private String stuName;
    private Date birthday;
    private String address;
    private String photo;//头像

}
```

3. 编写dao层及映射文件

   StudentDao:

```
package com.hs.dao;

import com.hs.pojo.Student;

import java.util.List;

public interface StudentDao {

    //查询所有
    public List<Student> queryAllStudents();
}
```

   StudentMapper.xml:

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.hs.dao.StudentDao">
```

```
    <resultMap id="stuResultMap" type="Student">
        <id property="id" column="sid"/>
        <result property="stuName" column="sname"/>
        <result property="birthday" column="birthday"/>
        <result property="address" column="address"/>
        <result property="photo" column="photo"/>
    </resultMap>

    <!-- 查询所有 -->
    <select id="queryAllStudents" resultMap="stuResultMap">
        select * from tb_stu
    </select>
</mapper>
```

4. 编写service层及实现类

StudentService:

```
package com.hs.service;

import com.hs.pojo.Student;

import java.util.List;

public interface StudentService {
    public List<Student> queryAllStudents();
}
```

StudentServiceImpl:

```
package com.hs.service.impl;

import com.hs.dao.StudentDao;
import com.hs.pojo.Student;
import com.hs.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
@Transactional
public class StudentServiceImpl implements StudentService {

    @Autowired
    private StudentDao studentDao;
    @Override
    public List<Student> queryAllStudents() {
        return studentDao.queryAllStudents();
    }
}
```

5. 编写Controller里面的查询所有信息的逻辑

   StudentController:

   @ResponseBody注解表示将返回值以json数据格式显示在浏览器上面

```
package com.hs.controller;


import com.hs.pojo.Student;
import com.hs.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.List;

@Controller
public class StudentController {

    @Autowired
    private StudentService studentService;

  /*  @RequestMapping("list")
    @ResponseBody
    public List<Student> listAllStudents(){
        return studentService.queryAllStudents();
    }*/

    @RequestMapping("list")
    public String listAllStudents(Model model){
        List<Student> students = studentService.queryAllStudents();
        model.addAttribute("list",students);
        return "stulist";
    }

}
```

6. 编写stulist.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: mumusan
  Date: 2021/5/17
  Time: 10:36
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<html>
<head>
```

```
    <title>Title</title>
</head>
<body>
<center>
    <h1>学生管理系统</h1>

    <table width="60%" heigh="40%">
        <tr>
            <td>编号</td>
            <td>学生姓名</td>
            <td>生日</td>
            <td>地址</td>
            <td>头像</td>
            <td><a href="">添加学生信息</a></td>

        </tr>

        <c:forEach items="${list}" var="s">
            <tr>
                <td>${s.id}</td>
                <td>${s.stuName}</td>
                <td>
                    <fmt:formatDate value="${s.birthday}" pattern="yyyy-MM-dd">
</fmt:formatDate>
                </td>
                <td>${s.address}</td>
                <td> <img src="${s.photo}" width="150px" height="150px"/></td>
                <td>
                    <a href="">修改</a>
                    <a href="">删除</a>
                </td>

            </tr>

        </c:forEach>
    </table>
</center>
</body>
</html>
```

7. 浏览器访问测试

   将登陆成功之后路径重定向到list

```
if(htUser!=null){
    return "redirect:list";
}
```

直接访问登录页面，登录成功之后就自动跳转到列表页面

如果在web.xml里面配上欢迎页，tomcat启动的时候将默认访问该页面

```
<welcome-file-list>
  <welcome-file>login.jsp</welcome-file>
</welcome-file-list>
```

demo3: 使用PageHelper实现分页功能

1. 引入PageHelper分页插件依赖

```
<!-- pagehelper分页插件 -->
<!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper -->
<dependency>
  <groupId>com.github.pagehelper</groupId>
  <artifactId>pagehelper</artifactId>
  <version>5.1.2</version>
</dependency>
```

2. 在mybatis配置文件里面配置分页的拦截器

   mybatis-config.xml

```
<!-- 配置分页插件 -->
   <plugins>
       <plugin interceptor="com.github.pagehelper.PageInterceptor">
           <!-- 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库-->
           <property name="helperDialect" value="mysql"/>
           <property name="reasonable" value="true"/>
       </plugin>
   </plugins>
```

3. 编写service接口及实现类分页的方法

   StudentService

```
//分页查询
public PageInfo<Student> queryStudentsByPage(int currentPage, int pageSize);
```

   StudentServiceImpl

```
@Override
public PageInfo<Student> queryStudentsByPage(int currentPage, int pageSize) {
    //传入参数：当前页和每页条数
    PageHelper.startPage(currentPage,pageSize);
    List<Student> students = studentDao.queryAllStudents();
    //通过包装获取分页所需的其他值
    PageInfo<Student> pageInfo = new PageInfo<>(students);
    return pageInfo;
}
```

4. 编写分页的controller

   StudentController:

```
@RequestMapping("listpage")
public String listAllStudents(Integer currentPage,Integer pageSize,Model model){
    if(currentPage==null){
        currentPage=1;
    }
    if(pageSize==null){
        pageSize=3;
    }
    PageInfo<Student> page = studentService.queryStudentsByPage(currentPage, pageSize);

    model.addAttribute("page",page);
    return "stulistpage";
}
```

5. 编写前端的分页页面

   stulistpage.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false"
isErrorPage="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<html>
<head>
    <title>Title</title>
</head>
<body>
欢迎${sessionScope.user.username}登陆
<center>
    <h1>学生管理系统</h1>

    <table width="60%" heigh="40%">
        <tr>
            <td>编号</td>
            <td>学生姓名</td>
            <td>生日</td>
            <td>地址</td>
            <td>头像</td>
            <td><a href="toadd">添加学生信息</a></td>
```

```
        </tr>

        <c:forEach items="${page.list}" var="s">
            <tr>
                <td>${s.id}</td>
                <td>${s.stuName}</td>
                <td>
                    <fmt:formatDate value="${s.birthday}" pattern="yyyy-MM-dd">
</fmt:formatDate>
                </td>
                <td>${s.address}</td>
                <td>
                    <img src="show?path=${s.photo}" width="150px" height="150px"/>
                </td>
                <td>
                    <a href="load?id=${s.id}">修改</a>
                    <a href="delete?id=${s.id}">删除</a>
                    <a href="download?path=${s.photo}">删除</a>
                </td>

            </tr>

        </c:forEach>
    </table>
    <input type="button" onclick="toFirst()" value="首页">
    <input type="button" onclick="toPrev()" value="上一页">
    当前页${page.pageNum}|${page.pages}总页数
    <input type="button" onclick="toNext()" value="下一页">
    <input type="button" onclick="toLast()" value="尾页">
    每页显示<input type="text" size="2" id="pageNo" value="${page.pageSize}">条记录
</center>
</body>
<script>

    function toFirst() {
        var pageSize= document.getElementById("pageNo").value;
        location.href="listpage?currentPage=1&pageSize="+pageSize;
    }
    function toPrev() {
        var pageSize= document.getElementById("pageNo").value;
        location.href="listpage?currentPage=${page.prePage}&pageSize="+pageSize;
    }
    function toNext() {
        var pageSize= document.getElementById("pageNo").value;
        location.href="listpage?currentPage=${page.nextPage}&pageSize="+pageSize;
    }
    function toLast() {
        var pageSize= document.getElementById("pageNo").value;
        location.href="listpage?currentPage=${page.lastPage}&pageSize="+pageSize;
    }
</script>
</html>
```

6. 修改登录成功之后跳转到分页的逻辑

   登录成功之后跳转到分页的逻辑，并且登录成功之后吧登陆的用户信息c存到HttpSession对象里面，进而可以在首页展示登录用户的信息

   UserController

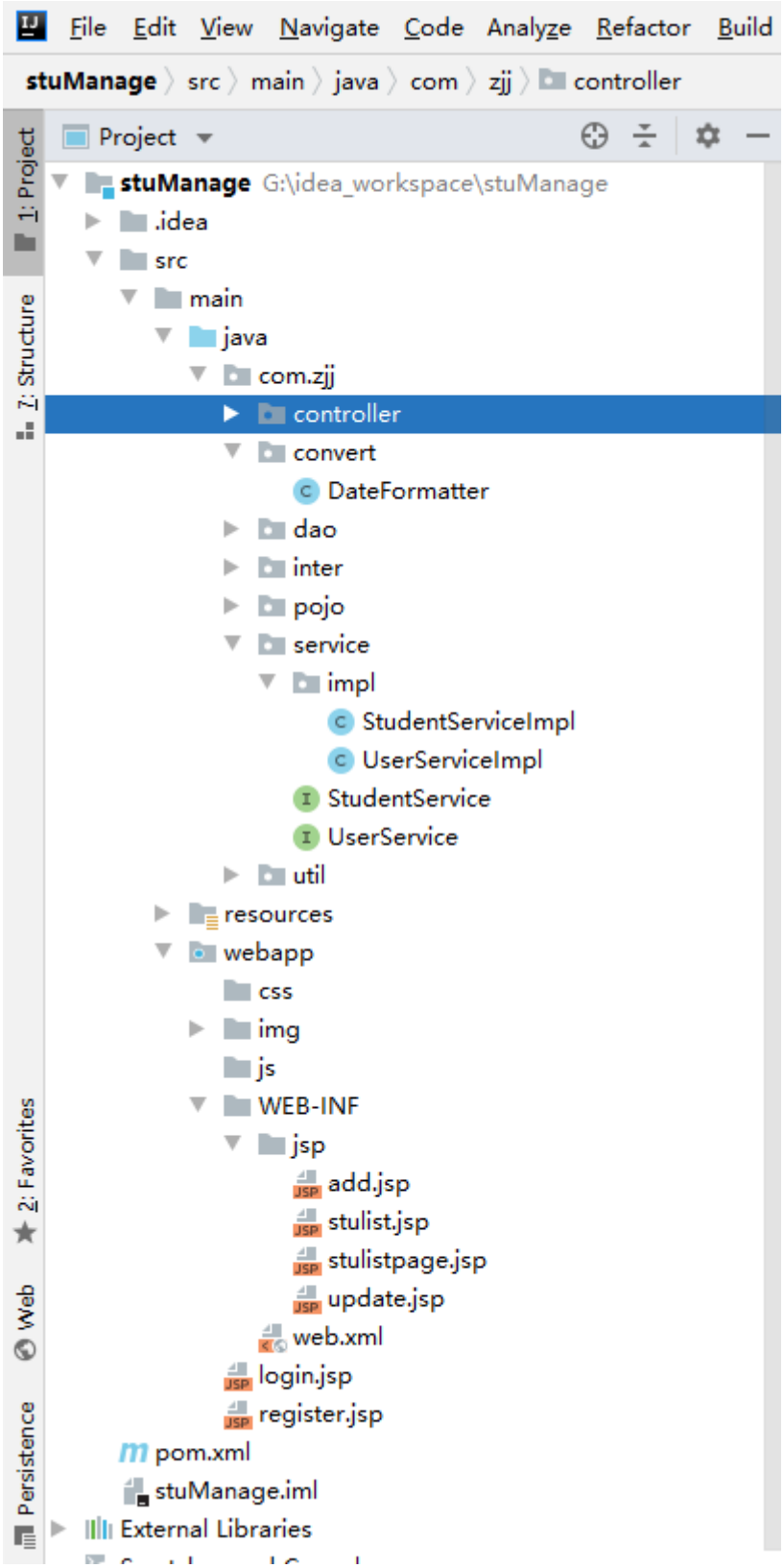
```
@RequestMapping("/login")
public String login(User user, HttpServletRequest request){
    User htUser = userService.login(user);
    //判断是否登录成功
    if(htUser!=null){
        HttpSession session = request.getSession();
        session.setAttribute("user",htUser);
        return "redirect:listpage";
    }
    //return "redirect:login.jsp";  重定向
   // return "forward:login.jsp";  请求转发
    return "redirect:login.jsp";
}
```

7. 浏览器访问测试

   http://localhost:8080/stuManage_war/login.jsp

demo4: 实现学生信息的增删改查功能

项目目录结构：


File Edit View Navigate Code Analyze Refactor Build
stuManage src main java com zjj controller
Project
stuManage G:\idea_workspace\stuManage
.idea
src
main
java
com.zjj
controller
convert
DateFormatter
dao
inter
pojo
service
impl
StudentServiceImpl
UserServiceImpl
StudentService
UserService
util
resources
webapp
css
img
js
WEB-INF
jsp
add.jsp
stulist.jsp
stulistpage.jsp
update.jsp
web.xml
login.jsp
register.jsp
pom.xml
stuManage.iml
External Libraries
1: Project
7: Structure
2: Favorites
Web
Persistence

思路：

添加：

由于add.jsp位于WEB-INF目录底下，WEB-INF下的文件内容客户端浏览器无法直接访问，所以需要经过后台中转下

```
<a href="toadd">添加学生信息</a>------>toAdd的后台controller------->add.jsp
----->用户输入添加数据，点击添加按钮，跳转到后台add的controller------>
添加成功，回到listpage分页列表;失败，回到add.jsp继续添加
```

删除:

根据id来删除该行的数据信息

```
<a href="delete?id=${s.id}">删除</a>------> delete 删除的后台controller
------->删除成功，回到listpage分页列表
```

修改:

修改功能需要用户再原数据的基础之上吧原数据修改成新数据，所以需要先获取原数据，获取原数据的过程就是查询单个的过程，所以修改时需要先根据id来查询单个

```
<a href="load?id=${s.id}">修改</a>------>load 后台查询单个------>回到update.jsp 原数据回显到修改页面
-------->    点击修改按钮，实现修改功能-------->update 后台修改功能------>修改成功，回到listpage分页列表
```

步骤:

1. dao层及映射文件配置

   StudentDao:

   ```
   package com.hs.dao;

   import com.hs.pojo.Student;

   import java.util.List;

   public interface StudentDao {

       //查询所有
       public List<Student> queryAllStudents();
   ```

```
    //查询单个
    public Student queryStudentById(int id);
    //添加学生
    public int addStudent(Student student);
    //修改学生
    public int updateStudent(Student student);
    //删除学生
    public int deleteStudentById(int id);
}
```

StudentMapper.xml:

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.hs.dao.StudentDao">

    <resultMap id="stuResultMap" type="Student">
        <id property="id" column="sid"/>
        <result property="stuName" column="sname"/>
        <result property="birthday" column="birthday"/>
        <result property="address" column="address"/>
        <result property="photo" column="photo"/>
    </resultMap>

    <!-- 查询所有 -->
    <select id="queryAllStudents" resultMap="stuResultMap">
        select * from tb_stu
    </select>

    <insert id="addStudent" parameterType="Student">
        INSERT INTO tb_stu (sname, birthday, address, photo) VALUES (#{stuName},#
{birthday},#{address},#{photo});
    </insert>

    <select id="queryStudentById" parameterType="int" resultMap="stuResultMap">
        select * from tb_stu where sid=#{id}
    </select>

    <update id="updateStudent" parameterType="Student">
        UPDATE tb_stu SET sname=#{stuName}, birthday=#{birthday}, address=#{address},
photo=#{photo} WHERE sid=#{id}
    </update>

    <delete id="deleteStudentById" parameterType="int">
        delete from tb_stu where sid=#{id}
    </delete>
</mapper>
```

2. service层及实现类代码

StudentService:

```
package com.hs.service;

import com.github.pagehelper.PageInfo;
import com.hs.pojo.Student;

import java.util.List;

public interface StudentService {
    //查询所有
    public List<Student> queryAllStudents();
    //分页查询
    public PageInfo<Student> queryStudentsByPage(int currentPage, int pageSize);

    //查询单个
    public Student queryStudentById(int id);
    //添加学生
    public boolean addStudent(Student student);
    //修改学生
    public boolean updateStudent(Student student);
    //删除学生
    public boolean deleteStudentById(int id);
}
```

StudentServiceImpl:

```
package com.hs.service.impl;

import com.github.pagehelper.PageHelper;
import com.github.pagehelper.PageInfo;
import com.hs.dao.StudentDao;
import com.hs.pojo.Student;
import com.hs.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
@Transactional
public class StudentServiceImpl implements StudentService {

    @Autowired
    private StudentDao studentDao;
    @Override
    public List<Student> queryAllStudents() {
        return studentDao.queryAllStudents();
    }

    @Override
    public PageInfo<Student> queryStudentsByPage(int currentPage, int pageSize) {
        //传入参数：当前页和每页条数
```

```
        PageHelper.startPage(currentPage,pageSize);
        List<Student> students = studentDao.queryAllStudents();
        //通过包装获取分页所需的其他值
        PageInfo<Student> pageInfo = new PageInfo<>(students);


        return pageInfo;
    }

    @Override
    public Student queryStudentById(int id) {
        return studentDao.queryStudentById(id);
    }

    @Override
    public boolean addStudent(Student student) {
        int row = studentDao.addStudent(student);
        if(row>0){
            return true;
        }
        return false;
    }

    @Override
    public boolean updateStudent(Student student) {
        int row = studentDao.updateStudent(student);
        if(row>0){
            return true;
        }
        return false;
    }

    @Override
    public boolean deleteStudentById(int id) {
        int row = studentDao.deleteStudentById(id);
        if(row>0){
            return true;
        }
        return false;
    }
}
```

3. controller层代码

   StudentController:

```
package com.hs.controller;


import com.github.pagehelper.PageInfo;
import com.hs.pojo.Student;
import com.hs.service.StudentService;
import com.hs.util.LoadUtil;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.List;

@Controller
public class StudentController {

    @Autowired
    private StudentService studentService;

  /*  @RequestMapping("list")
    @ResponseBody
    public List<Student> listAllStudents(){
        return studentService.queryAllStudents();
    }*/

    @RequestMapping("list")
    public String listAllStudents(Model model){
        List<Student> students = studentService.queryAllStudents();
        model.addAttribute("list",students);
        return "forward:/stulist.jsp";
    }

    @RequestMapping("listpage")
    public String listAllStudents(Integer currentPage,Integer pageSize,Model model){
        if(currentPage==null){
            currentPage=1;
        }
        if(pageSize==null){
            pageSize=3;
        }
        PageInfo<Student> page = studentService.queryStudentsByPage(currentPage,
pageSize);

        model.addAttribute("page",page);
        return "stulistpage";
    }

    @RequestMapping("add")
    public String addStudent(Student student, MultipartFile img){
        String photo = LoadUtil.uploadPhoto(img);
        student.setPhoto(photo);
        boolean flag = studentService.addStudent(student);
        if(flag){
            return "redirect:listpage";
        }
        return "redirect:toadd";
```

```
    }


    @RequestMapping("toadd")
    public String toAdd(){

        return "add";
    }

    @RequestMapping("update")
    public String updateStudent(Student student){
        boolean flag = studentService.updateStudent(student);

        return "redirect:listpage";
    }

    @RequestMapping("delete")
    public String deleteStudent(int id){
        studentService.deleteStudentById(id);
        return "redirect:listpage";
    }
    @RequestMapping("load")
    public String loadStudent(int id, Model model){
        Student student = studentService.queryStudentById(id);
        model.addAttribute("s",student);
        return "update";
    }
}
```

4. 日期类型的格式转换器代码编写及配置

   由于add或update是有日期类型的数据，而springmvc无法直接取到日期类型的数据，所以需要自定义一个日期类型的格式转换器

   DateFormatter:

```
package com.hs.convert;

import org.springframework.format.Formatter;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

public class DateFormatter implements Formatter<Date> {
    //日期格式化对象
    private SimpleDateFormat dateFormat;

    public SimpleDateFormat getDateFormat() {
        return dateFormat;
    }

    public void setDateFormat(SimpleDateFormat dateFormat) {
```

```
        this.dateFormat = dateFormat;
    }

    //解析
    @Override
    public Date parse(String s, Locale locale) throws ParseException {
        return dateFormat.parse(s);
    }
    //显示
    @Override
    public String print(Date date, Locale locale) {
        return dateFormat.format(date);
    }
}
```

spring-mvc.xml:

```
<!-- 注解驱动 表示对springmvc相关注解的支持 -->
<!-- json 转换器 -->
<mvc:annotation-driven conversion-service="conversionService">
    ......
</mvc:annotation-driven>

<!--转换器-->
<bean id="conversionService"
class="org.springframework.format.support.FormattingConversionServiceFactoryBean">

    <!-- 配置自定义格式转换器-->
    <property name="formatters">
        <list>
            <bean class="com.hs.convert.DateFormatter">
                <property name="dateFormat" value="yyyy-MM-dd"/>
            </bean>
        </list>

    </property>
</bean>
```

5. 前端页面

stulistpage.jsp：

```
<table width="60%" heigh="40%">
    <tr>
        <td>编号</td>
        <td>学生姓名</td>
        <td>生日</td>
        <td>地址</td>
        <td>头像</td>
        <td><a href="toadd">添加学生信息</a></td>

    </tr>
```

```
        <c:forEach items="${page.list}" var="s">
            <tr>
                <td>${s.id}</td>
                <td>${s.stuName}</td>
                <td>
                    <fmt:formatDate value="${s.birthday}" pattern="yyyy-MM-dd">
</fmt:formatDate>
                </td>
                <td>${s.address}</td>
                <td>
                    <img src="${s.photo}" width="150px" height="150px"/>
                </td>
                <td>
                    <a href="load?id=${s.id}">修改</a>
                    <a href="delete?id=${s.id}">删除</a>

                </td>

            </tr>

        </c:forEach>
    </table>
```

add.jsp:

```
    <%@ page contentType="text/html;charset=UTF-8" language="java" %>
    <html>
    <head>
        <title>Title</title>
    </head>
    <body>

    <center>
        <h1>添加学生</h1>
        <form action="add" method="post">

            <table>
                <tr>
                    <td>姓名</td>
                    <td><input type="text" name="stuName"></td>
                </tr>

                <tr>
                    <td>生日</td>
                    <td><input type="date" name="birthday"></td>
                </tr>
                <tr>
                    <td>地址</td>
                    <td><input type="text" name="address"></td>
                </tr>
                <tr>
                    <td>头像</td>
                    <td><input type="text" name="photo"></td>
```

```
            </tr>

            <tr>
                <td><input type="submit" value="添加学生"></td>
                <td><input type="reset" value="重置"></td>
            </tr>
        </table>
    </form>
</center>
</body>
</html>
```

update.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

<center>
    <h1>修改学生</h1>
    <form action="update" method="post">

        <table>
            <tr>
                <td>编号</td>
                <td><input type="text" name="id" value="${s.id}" readonly></td>
            </tr>
            <tr>
                <td>姓名</td>
                <td><input type="text" name="stuName" value="${s.stuName}"></td>
            </tr>

            <tr>
                <td>生日</td>
                <td><input type="date" name="birthday" value="${s.birthday}"></td>
            </tr>
            <tr>
                <td>地址</td>
                <td><input type="text" name="address" value="${s.address}"></td>
            </tr>
            <tr>
                <td>头像</td>
                <td><input type="text" name="photo" value="${s.photo}"></td>
            </tr>
            <tr>
```

```
                <td><input type="submit" value="修改学生"></td>
                <td><input type="reset" value="重置"></td>
            </tr>
        </table>
    </form>

</center>



</body>
</html>
```

6. 浏览器访问测试效果

demo5: 使用拦截器实现用户未经登录不能访问主逻辑功能

1. 创建LoginInterceptor，实现HandlerInterceptor接口

   3个方法，拦截的时机不同：

   boolean preHandle(): 方法执行之前拦截; 返回值true表示放行，false表示拦截

   void postHandle(): 方法接收请求之后，解析视图之前拦截

   void afterCompletion(): 方法执行完毕，视图渲染结束之后执行

```
package com.hs.inter;

import com.hs.pojo.User;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class LoginInterceptor implements HandlerInterceptor {
    //方法执行之前拦截
    @Override
    public boolean preHandle(HttpServletRequest httpServletRequest, HttpServletResponse
httpServletResponse, Object o) throws Exception {
        HttpSession session = httpServletRequest.getSession();
        User user = (User) session.getAttribute("user");
        //true 放行 false拦截
        if(user!=null){
            return true;
        }
        httpServletResponse.sendRedirect("login.jsp");
        return false;
```

```
    }
    //方法接收请求之后，解析视图之前拦截
    @Override
    public void postHandle(HttpServletRequest httpServletRequest, HttpServletResponse
httpServletResponse, Object o, ModelAndView modelAndView) throws Exception {

    }
    //方法执行完毕，视图渲染结束之后执行
    @Override
    public void afterCompletion(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, Object o, Exception e) throws Exception {

    }
}
```

2. 在springmvc配置文件里面进行拦截器的配置

   spring-mvc.xml

```
<!-- 配置类似于全局的拦截器  如果项目中用到拦截器，需要配此项，如果没用的到拦截器，不能配，否则会报错-->
<mvc:interceptors>
    <mvc:interceptor>

        <mvc:mapping path="/**"/>

        <mvc:exclude-mapping path="/login"/>
        <bean class="com.hs.inter.LoginInterceptor"></bean>
   </mvc:interceptor>
</mvc:interceptors>
```

   说明：

   /** 表示拦截所有

   表示放行的路径

   这个路径要和自己的拦截器路径保持一致

   3. 测试访问其他的逻辑，如果未经登录，会被拦截，回到login.jsp

demo6： 实现用户头像的上传下载功能：

1. 上传下载的工具类

   为了方便起见，直接将相关方法封装到了一个工具类里面了，静态方法，直接调用方便

   LoadUtil：

```
package com.hs.util;

import org.springframework.web.multipart.MultipartFile;
```

```
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;

public class LoadUtil {
   //上传图片
   public static String uploadPhoto(MultipartFile loadimg){
      //获取绝对路径
      String path ="D:/upload";
      File f=new File(path);
      //如果不存在，直接创建
      if(!f.exists()){
         f.mkdirs();
      }
      //获取图片名称
      String filename = loadimg.getOriginalFilename();
      //拼接的图片路径
      String filepath=path+"/"+filename;
      File file = new File(filepath);
      //上传图片
      try {
         loadimg.transferTo(file);
      } catch (IllegalStateException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      } catch (IOException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }
      return filepath;
   }
   //回显图片
   public static void showPhoto(String photo,HttpServletResponse response){
      //获取图片的当前路径  放入读
       FileInputStream fis=null;
      //用response  获取一个写对象的流
      ServletOutputStream os=null;
      try {
         fis = new FileInputStream(photo);
          os = response.getOutputStream();
          //提高读写的速度
          byte[] b=new byte[1024];
          //边读边写
          while(fis.read(b)!=-1){
             os.write(b);
          }
      } catch (FileNotFoundException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      } catch (IOException e) {
         // TODO Auto-generated catch block
```

```
            e.printStackTrace();
        }finally{
            try {
                if(os!=null){
                    os.close();
                }
                if(fis!=null){
                    fis.close();
                }
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }

    }
    //下载
    public static void downLoad(String filepath,HttpServletRequest request,
HttpServletResponse response){
        //设置文件的MiMe类型

response.setContentType(request.getSession().getServletContext().getMimeType(filepath))
;
        //设置content-dispsition
        response.setHeader("Content-Disposition", "attachment;filename="+filepath);
        //读取目标文件，通过response将目标文件写到客户
        try {
            //读取文件
            InputStream  in = new FileInputStream(filepath);
            OutputStream out=response.getOutputStream();
            //写文件
            byte[] b=new byte[1024];
            while(in.read(b)!=-1){
                out.write(b);
            }
            in.close();
            out.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

2. add.jsp页面改造

   add.jsp

   注意：

   method="post" enctype="multipart/form-data"： method必须post请求， enctype="multipart/form-data"表示处理二进制文件

type="file": 图片使用文件上传框

```
<form action="add" method="post" enctype="multipart/form-data">

    <table>
       .......
       <%-- <tr>
            <td>头像</td>
            <td><input type="text" name="photo"></td>
        </tr>--%>
        <tr>
            <td>请上传头像: </td>
            <td><input type="file" name="img"></td>

        </tr>
        <tr>
            <td>&lt;input type="submit" value="添加学生"&gt;&lt;/td&gt;
            <td><input type="reset" value="重置"></td>
        </tr>




    </table>
</form>
```

3. 后台controller实现上传下载功能

StudentController:

```
//添加
@RequestMapping("add")
public String addStudent(Student student, MultipartFile img){
    String photo = LoadUtil.uploadPhoto(img);
    student.setPhoto(photo);
    boolean flag = studentService.addStudent(student);
    if(flag){
    return "redirect:listpage";
    }
    return "redirect:toadd";
}
//显示头像
@RequestMapping("show")
public void showPhoto(String path, HttpServletResponse response){
    LoadUtil.showPhoto(path,response);
}
//下载
@RequestMapping("download")
public void download(String path, HttpServletRequest request,HttpServletResponse
response){
    LoadUtil.downLoad(path, request, response);

}
```

4. listpage.jsp 显示图片

   由于某些情况下图片不能直接显示，所以需要显示图片的逻辑，就是controller里面的showPhoto()方法

```
<c:forEach items="${page.list}" var="s">
    <tr>
       .........
        <td>
            <img src="show?path=${s.photo}" width="150px" height="150px"/>
        </td>
        <td>
            <a href="load?id=${s.id}">修改</a>
            <a href="delete?id=${s.id}">删除</a>
            <a href="download?path=${s.photo}">下载图片</a>
        </td>

    </tr>

</c:forEach>
```

5. 测试效果

demo7： 实现登录页面的国际化显示：

1. 在resources目录下新建两个properties文件，文件名及内容如下

   message_en.properties：

```
username=username
password=password
login=login
register=register
title=Login Page
```

   message_zh_CN.properties：存储的是中文(unicode码)：

```
username=\u7528\u6237\u540d
password=\u5bc6\u7801
login=\u767b\u9646
register=\u7acb\u5373\u6ce8\u518c
title=\u767b\u9646\u9875\u9762
```

   不会转的使用这个工具类：

```
public class TestBianMa {

    public static void main(String[] args) {
        String uname = "登陆页面";
```

```
        for (int i = 0; i < uname.length(); i++) {
            char unamechar = uname.charAt(i);
            System.out.print(gbEncoding(String.valueOf(unamechar)));
        }

    }

    /**
     * 把中文转成Unicode编码
     *
     * @param gbString
     * @return
     */
    private static String gbEncoding(final String gbString) {
        char[] utfBytes = gbString.toCharArray();
        String unicodeBytes = "";
        for (int byteIndex = 0; byteIndex < utfBytes.length; byteIndex++) {
            String hexB = Integer.toHexString(utfBytes[byteIndex]);
            if (hexB.length() <= 2) {
                hexB = "00" + hexB;
            }
            unicodeBytes = unicodeBytes + "\\u" + hexB;
        }
        return unicodeBytes;
    }

}
```

2. 在springmvc xml中配置国际化信息

   默认中文

```
<!--国际化配置-->
<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="/message"/>
</bean>
<bean id="cookieLocaleResolver"
class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
    <property name="defaultLocale" value="zh_CN"/>
    <property name="cookieName" value="language"/>
</bean>
```

3. jsp中登录页面

   login.jsp

注意：需要引入spring的标签库：<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<html>
<head>
```

```
    <title>Title</title>
</head>
<body>
<center>
    <h1>
        <spring:message code="title"/>
    </h1>
    <form action="login" method="post">

        <table>
            <tr>
                <td> <spring:message code="username"/></td>
                <td><input type="text" name="username"></td>
            </tr>
            <tr>
                <td> <spring:message code="password"/></td>
                <td><input type="password" name="password"></td>
            </tr>
            <tr>
                <td><input type="submit" value="<spring:message code='login'/>"></td>
                <td><a href="register.jsp"><spring:message code="register"/></a></td>
            </tr>
        </table>

    </form>
</center>

</body>
</html>
```

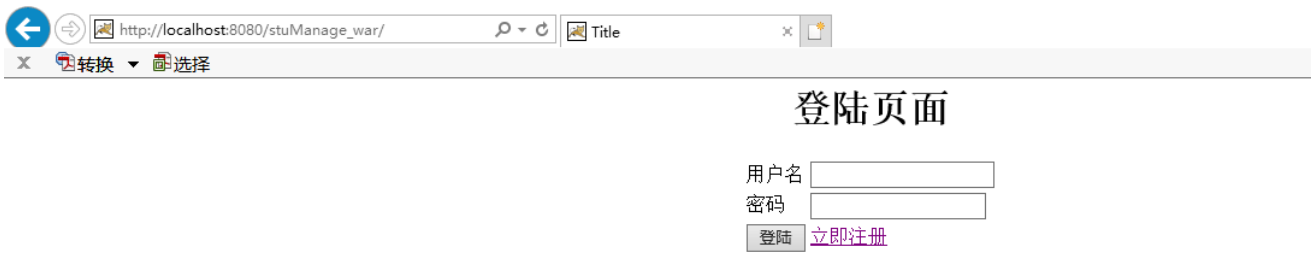
4. 一般中文是默认语言，所以需要设置英文成默认语言，以win10为例

   设置 ---》 时间和语言 ---》左侧选择 区域和语言

没有英文的就需要添加语言

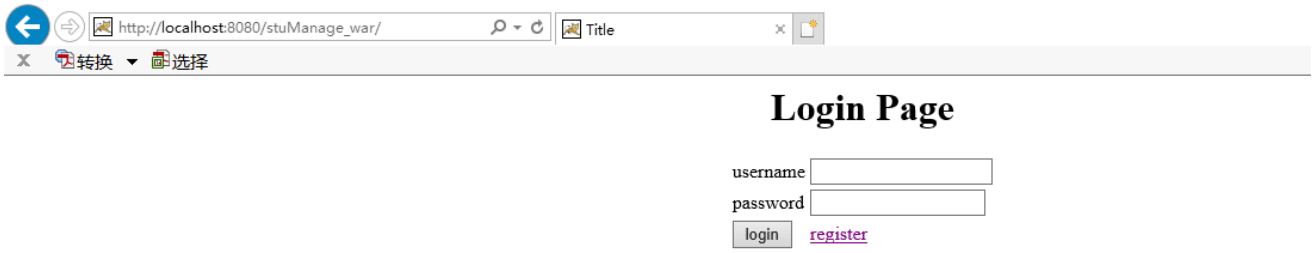
到此就实现了国际化，效果如下：

中文状态下：



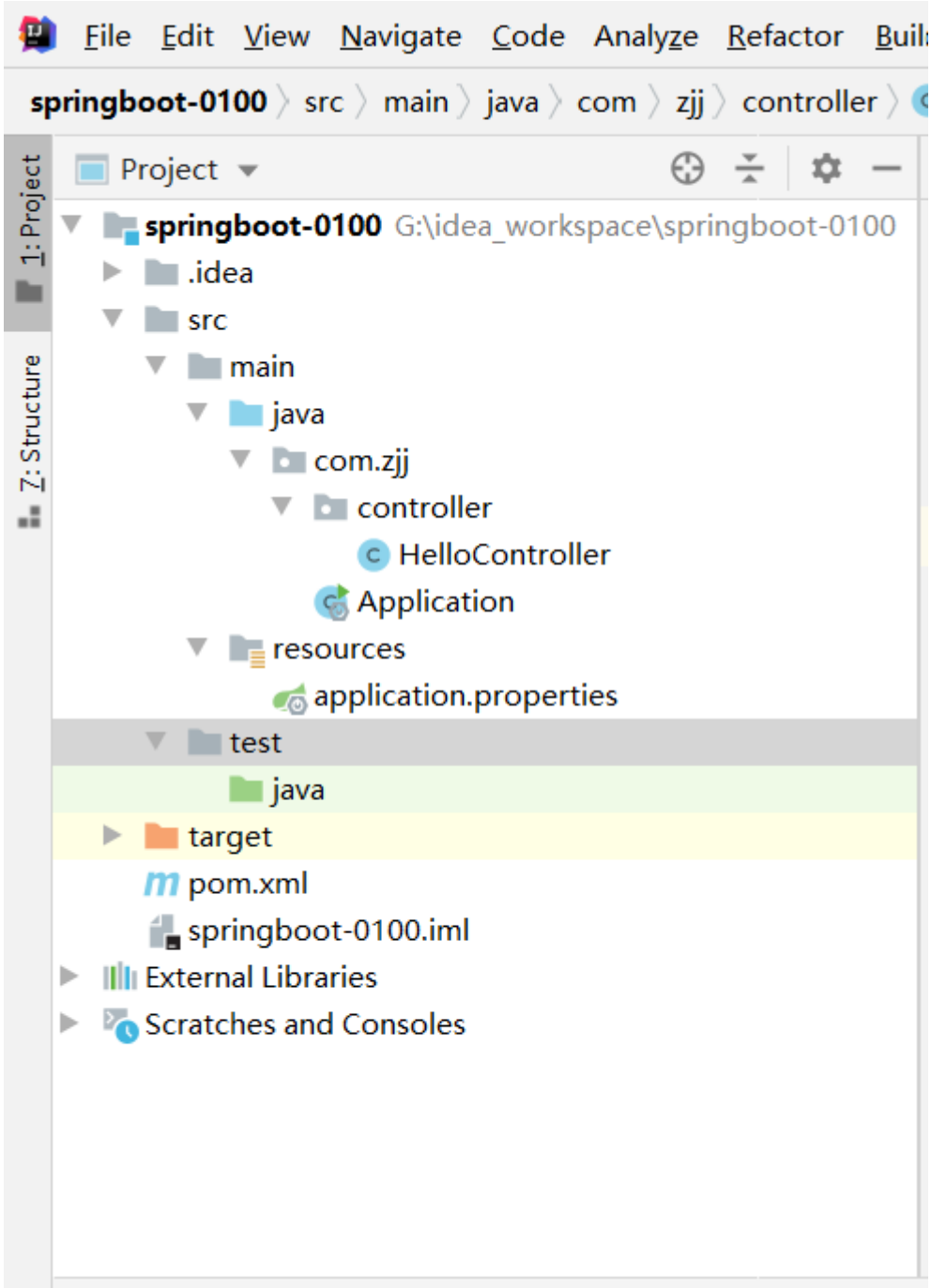
切换英文：

刷新浏览器：



SpringBoot环境搭建：

1.项目目录结构



2.新建maven工程

注意： maven project: 不是maven web project

3.添加依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
    <groupId>org.example</groupId>
    <artifactId>springboot-0100</artifactId>
    <version>1.0-SNAPSHOT</version>

    <parent>
        <artifactId>spring-boot-starter-parent</artifactId>
        <groupId>org.springframework.boot</groupId>
        <version>2.1.4.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>


</project>
```

4. 编写application.properties文件

在resources根目录底下，文件名固定，不能改

application.properties： 是SpringBoot的全局配置文件

```
server.port=8181
```

5. 编写main方法

Application：

```
package com.hs;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class,args);
    }
}
```

6. 编写controller

HelloController：

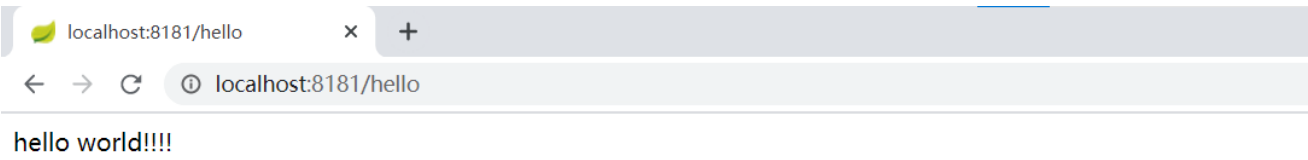
```
package com.hs.controller;


import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @RequestMapping("hello")
    public String hello(){
        return "hello world!!!!";
    }
}
```

7. 启动main方法

8.浏览器访问测试

http://localhost:8181/hello

运行效果：



注意：

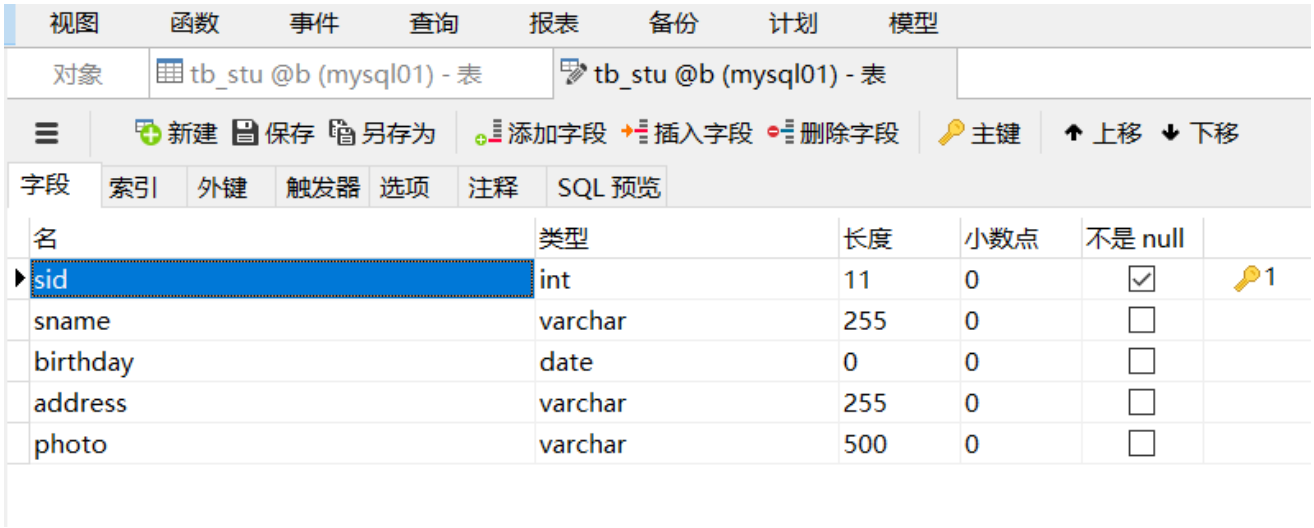
浏览器访问的端口号和配置文件的端口号一致，

main方法所在的类必须在所有组件的父包里面，这样它会自动扫描

springboot里面内置了tomcat；直接启动main方法即启动tomcat服务器

SpringBoot+MyBatis实现增删改查

1. 设计数据库表



名	类型	长度	小数点	不是 null	
sid	int	11	0	☑	1
sname	varchar	255	0	☐	
birthday	date	0	0	☐	
address	varchar	255	0	☐	
photo	varchar	500	0	☐	

2. 新建maven project

注意：不是maven web project,不需要勾选那个复选框

3. pom文件添加依赖

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.bw</groupId>
  <artifactId>stuManage-boot</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>


  <parent>
    <artifactId>spring-boot-starter-parent</artifactId>
    <groupId>org.springframework.boot</groupId>
    <version>2.1.4.RELEASE</version>
  </parent>


  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
    </dependency>

    <dependency>
      <groupId>org.mybatis.spring.boot</groupId>
      <artifactId>mybatis-spring-boot-starter</artifactId>
      <version>2.1.3</version>
    </dependency>

    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
    </dependency>

    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
    </dependency>
  </dependencies>

</project>
```

```
    <artifactId>spring-boot-starter-parent</artifactId>
    <groupId>org.springframework.boot</groupId>
    <version>2.1.4.RELEASE</version>
</parent>
```

表示父工程是个SpringBoot项目，继承它自己就成为一个SpringBoot工程了

4. 配置项目配置信息 application.yml

配置文件有两种格式： properties和yml

properties:里面格式是键值对

yml: 一定要注意缩进；相同层次的缩进必须保持相同，否则会有语法问题

```
server:
  port: 8989

spring:
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/b?
characterEncoding=utf8&useSSL=false&serverTimezone=UTC&rewriteBatchedStatements=true
    username: root
    password: 123456

mybatis:
```

```
  type-aliases-package: com.hs.pojo
  mapper-locations: classpath:mapper/*Mapper.xml
logging:
  level:
    com:
      zjj:
        dao: debug
```

5. 编写main方法

SpringBoot项目的启动类，必须加上注解@SpringBootApplication

@MapperScan("com.hs.dao")： 表示扫描dao层

```
package com.hs;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@MapperScan("com.hs.dao")
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class,args);
    }
}
```

6. 编写pojo实体类

```
package com.hs.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.format.annotation.DateTimeFormat;

import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {
    private Integer id;
    private String stuName;

    @DateTimeFormat(pattern="yyyy-MM-dd")
    private Date birthday;
    private String address;
```

```
    private String photo;//头像

}
```

7. 编写dao层代码及映射文件

接口上面必须加注解@Mapper

```
package com.hs.dao;

import com.hs.pojo.Student;
import org.apache.ibatis.annotations.Mapper;

import java.util.List;

@Mapper
public interface StudentDao {

    //查询所有
    public List<Student> queryAllStudents();

    public int addStudent(Student student);

    public int deleteStudentById(Integer id);

    public int updateStudent(Student student);

    public Student queryOneStudentById(Integer id);
}
```

StudentMapper.xml映射文件，位于resources/mapper目录底下

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.hs.dao.StudentDao">

    <resultMap id="stuResultMap" type="Student">
        <id property="id" column="sid"/>
        <result property="stuName" column="sname"/>
        <result property="birthday" column="birthday"/>
        <result property="address" column="address"/>
        <result property="photo" column="photo"/>
    </resultMap>

    <!-- 查询所有 -->
    <select id="queryAllStudents" resultMap="stuResultMap">
        select * from tb_stu
    </select>

    <select id="queryOneStudentById" parameterType="int" resultMap="stuResultMap">
```

```
        select * from tb_stu where sid=#{id}
    </select>

    <insert id="addStudent" parameterType="Student">
        INSERT INTO `tb_stu` (`sid`, `sname`, `birthday`, `address`, `photo`) VALUES (0, #
{stuName}, #{birthday}, #{address}, #{photo})
    </insert>

    <update id="updateStudent" parameterType="Student">
        UPDATE tb_stu SET  `sname`=#{stuName},  `birthday`=#{birthday}, `address`=#
{address}, `photo`=#{photo} WHERE `sid`=#{id}
    </update>

    <delete id="deleteStudentById" parameterType="int">
        delete from tb_stu where sid=#{id}
    </delete>
</mapper>
```

8. 编写service层代码及实现类

Service接口：

```
package com.hs.service;

import com.hs.pojo.Student;

import java.util.List;

public interface StudentService {
    public List<Student> queryAllStudents();

    public boolean addStudent(Student student);

    public boolean deleteStudentById(Integer id);

    public boolean updateStudent(Student student);

    public Student queryOneStudentById(Integer id);
}
```

Service实现类： 注解上面的注解 @Service @Transactional

```
package com.hs.service.impl;

import com.hs.dao.StudentDao;
import com.hs.pojo.Student;
import com.hs.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
```

```
import java.util.List;

@Service
@Transactional
public class StudentServiceImpl implements StudentService {

    @Autowired
    private StudentDao studentDao;
    @Override
    public List<Student> queryAllStudents() {
        return studentDao.queryAllStudents();
    }

    @Override
    public boolean addStudent(Student student) {
        int row = studentDao.addStudent(student);
        if(row>0){
            return true;
        }
        return false;
    }

    @Override
    public boolean deleteStudentById(Integer id) {
        int row = studentDao.deleteStudentById(id);
        if(row>0){
            return true;
        }
        return false;
    }

    @Override
    public boolean updateStudent(Student student) {
        int row = studentDao.updateStudent(student);
        if(row>0){
            return true;
        }
        return false;
    }


    @Override
    public Student queryOneStudentById(Integer id) {
        return studentDao.queryOneStudentById(id);
    }


}
```

9. 编写controller层代码及实现类

@RestController 表示返回json数据到浏览器上面

```
package com.hs.controller;


import com.hs.pojo.Student;
import com.hs.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class StudentController {

    @Autowired
    private StudentService studentService;

    @GetMapping
    public List<Student> listAllStudents(){
        return studentService.queryAllStudents();

    }

    @GetMapping("/{id}")
    public Student loadStudentById(@PathVariable int id){
        return studentService.queryOneStudentById(id);

    }

    @PostMapping
    public boolean addStudent(@RequestBody Student student){
       return studentService.addStudent(student);
    }

    @PutMapping
    public boolean updateStudent(@RequestBody Student student){
        return studentService.updateStudent(student);

    }

    @DeleteMapping("{id}")
    public boolean deleteStudentById(@PathVariable int id){
        return studentService.deleteStudentById(id);
    }

}
```

10. 启动main方法

SpringBoot内置Tomcat,启动main方法即启动tomcat

SpringBoot对jsp支持不好，一般SpringBoot项目使用前后端分离

11.使用PostMan访问测试效果

下载PostMan测试软件

Vue语法：

参考文档： https://www.runoob.com/vue2/vue-tutorial.html

环境： nodejs

前端开发工具： Visual Studio Code

自行下载安装：

nodejs安装步骤参考网上教程；测试nodejs是否 安装成功：node -v

VSC: 安装成功后 安装两个插件：

扩展 菜单搜索 Chinese Language 中文插件 Live Server服务器 安装即可

MVVM模型: Model View View Model

双向绑定：

vue-demo:

需要下载并引入vue.min.js vue的库文件

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <div id="app">

        <h1>vue语法</h1>
        <h1>您的生日是：{{
            new Date(birthday).getFullYear() + '-'+ new Date(birthday).getMonth()+ '-' +
new Date(birthday).getDay()
            }}
        </h1>
        <h1>您的生日是：{{birth}} </h1>

        欢迎您： {{name}}<br>
        <font color="red" v-text="name"></font>
```

```
<font color="red" v-html="name"></font>
num:{{num}}<br>
num: <input v-model="num"/><br>
num1: {{num1}}  num2: {{num2}}
请输入第一个数:  <input v-model="num1"/>
请输入第二个数:  <input v-model="num2"/>
和是:{{sum}}  {{add2()}}
求和:  <input type="button" value="求和" @click="add"/><br>
地址:<input v-model="person.address"/>
取对象属性:
name: {{person.name}}  age: {{person.age}} address: {{person.address}}<br>
取数组:
{{arr}}
循环遍历:
<ul v-for="a in arr">
    <li v-text="a"></li>
</ul>
<br>
<!-- a 表示某元素 i 表示下标-->
<ul v-for="(a,i) in arr">
    <li v-text="i"></li>
</ul>
<br>

<p v-if="num>num1">{{num}}</p>
<p v-if="num<=num1">{{num1}}</p>

<table border="1">

    <tr>
        <td>编号</td>
        <td>姓名</td>
        <td>年龄</td>
        <td><a href="#">添加</a></td>

    </tr>
    <tr v-for="s in list">
        <td>{{s.id}}</td>
        <td>{{s.name}}</td>
        <td>{{s.age}}</td>
        <td>
            <a href="#">删除</a>
            <a href="#">修改</a>

        </td>

    </tr>
</table>
```

```
    </div>

</body>
<script src="js/vue.min.js"></script>
<script>

    new Vue({
        el: "#app",
        data: {
            name: "<h1>张三</h1>",
            num: 23,
            num1:0,
            num2:0,
            sum:0,
            person:{
                name: "李四",
                age:23,
                address:"北京"
            },
            arr:["噢噢噢噢","嗯嗯呃呃","哈哈哈","啊啊啊"],

            list:[],
            birthday:1529032123201 // 毫秒值

        },
        methods: {
            add: function(){
                this.sum=parseInt(this.num1)+parseInt(this.num2);
            },
            add2: function(){
                return parseInt(this.num1)+parseInt(this.num2);
            }
        },
        //页面加载事件
        created () {
            this.list=[
                {id:'1',name:'王五',age:23},
                {id:'2',name:'赵柳',age:24},
                {id:'3',name:'张德',age:18}
            ]
        },
        computed:{
                    birth(){// 计算属性本质是一个方法，但是必须返回结果
                        const d = new Date(this.birthday);
                        return d.getFullYear() + "-" + d.getMonth() + "-" + d.getDay();
                    }
                }
    })
</script>
</html>
```

使用VUE搭建前端页面，实现前后端对接：

注意：

页面需要引入3个库文件：

vue的库文件， ajax的库文件， 日期处理的库文件

stulist: 列表页面

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <div id="app">
            <h1>学生列表</h1>

            <table >
                <tr>
                    <td>编号</td>
                    <td>学生姓名</td>
                    <td>生日</td>
                    <td>地址</td>
                    <td>头像</td>
                    <td><a href="add.html">添加学生</a></td>
                </tr>

                <tr v-for="s in list">
                    <td>{{s.id}}</td>
                    <td>{{s.stuName}}</td>
                    <td>
                        {{moment(s.birthday).format("YYYY-MM-DD")}}
                    </td>
                    <td>{{s.address}}</td>
                    <td>{{s.photo}}</td>
                    <td>

                        <input type="button" @click="update(s.id)" value="修改"/>
                        <input type="button" @click="del(s.id)" value="删除"/>
                    </td>
                </tr>
            </table>
    </div>
```

```
</body>
<script src="js/vue.min.js"></script>
<script src="js/axios.min.js"></script>
<script src="js/moment-with-locales.js"></script>
<script>
    new Vue({

        el: "#app",
        data: {
            id:0,
            list:[] //集合使用[]  对象使用{}
        },
        //初始化方法 被创建的时候执行的逻辑
        created () {
            this.getData();
        },
        methods: {
            getData: function(){
                axios.get("http://127.0.0.1:8989").then(
                  resp=>{

                      this.list=resp.data;
                      console.log(this.list);
                  }
                );
            },
            del: function(id){
                this.id=id;
                axios.delete("http://127.0.0.1:8989/"+this.id).then(
                  resp=>{

                    console.log(resp);
                    if(resp.data){
                        alert("删除成功");
                        location="stulist.html";
                    }else{
                        alert("删除失败");
                    }
                  }
                );
            },
            update:function(id){
                window.sessionStorage.setItem("id",id);
                location="update.html";
            }
        }

    })

</script>
</html>
```

add.html: 添加页面

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <div id="app">
            <h1>添加学生</h1>

            <table>
                <tr>
                    <td>学生姓名</td>
                    <td><input v-model="stu.stuName"/></td>

                </tr>
                <tr>
                    <td>生日</td>
                    <td><input type="date" v-model="stu.birthday"/></td>

                </tr>
                <tr>
                    <td>地址</td>
                    <td><input v-model="stu.address"/></td>

                </tr>
                <tr>
                    <td>头像</td>
                    <td><input v-model="stu.photo"/></td>

                </tr>

                <tr>
                    <td><input type="button" @click="fanhui" value="返回"/></td>
                    <td>
                        <input type="button" @click="chongzhi" value="重置"/>
                        <input type="button" @click="save" value="保存"/>

                    </td>

                </tr>


            </table>

    </div>
```

```
</body>
<script src="js/vue.min.js"></script>
<script src="js/axios.min.js"></script>
<script src="js/moment-with-locales.js"></script>
<script>
    new Vue({

        el: "#app",
        data:
            {

                stu: {
                        id: 0,
                        stuName: "",
                        birthday: moment(new Date()).format("YYYY-MM-DD"),
                        address:"",
                        photo:""
                }

            },

        methods: {


            fanhui: function(){
                location="stulist.html";
            },
            chongzhi:function(){
                location="add.html";
            },
            //保存方法
            save: function(){
                axios.post("http://127.0.0.1:8989",this.stu).then(
                  resp=>{
                      console.log(resp);
                      if(resp.data){
                          alert("添加成功");
                          location="stulist.html";
                      }else{
                          alert("添加失败");
                      }

                  }
                );
            }


        }




    })
```

```
</script>
</html>
```

update.html: 修改页面

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <div id="app">
            <h1>修改学生</h1>

            <table>
                <tr>
                    <td>学生编号</td>
                    <td><input v-model="stu.id" readonly/></td>
                </tr>
                <tr>
                    <td>学生姓名</td>
                    <td><input v-model="stu.stuName"/></td>

                </tr>
                <tr>
                    <td>生日</td>
                    <td><input type="date" v-model="stu.birthday"/></td>

                </tr>
                <tr>
                    <td>地址</td>
                    <td><input v-model="stu.address"/></td>

                </tr>
                <tr>
                    <td>头像</td>
                    <td><input v-model="stu.photo"/></td>

                </tr>

                <tr>
                    <td>
                        <input type="button" @click="update()" value="修改"/>
                    </td>
                    <td>
```

```
                        <input type="button" @click="fanhui" value="返回"/>
                    </td>

                </tr>


            </table>

    </div>

</body>
<script src="js/vue.min.js"></script>
<script src="js/axios.min.js"></script>
<script src="js/moment-with-locales.js"></script>
<script>
    new Vue({

        el: "#app",
        data:
            {

                stu: {
                        id: 0,
                        stuName: "",
                        birthday: moment(new Date()).format("YYYY-MM-DD"),
                        address:"",
                        photo:""
                }

            },

        created () {
           this.load();
        },

        methods: {


            fanhui: function(){
                location="stulist.html";
            },

            //修改方法
            update: function(){
                axios.put("http://127.0.0.1:8989",this.stu).then(
                  resp=>{
                      console.log(resp);
                      if(resp.data){
                          alert("修改成功");
                          location="stulist.html";
                      }else{
                          alert("修改失败");
                      }
```

```
                }
            );
        },
        load:function(){
            var id=window.sessionStorage.getItem("id");
            console.log("id: "+id);
            axios.get("http://127.0.0.1:8989/"+id).then(
              resp=>{
                  console.log(resp);
                  this.stu=resp.data;

                }
            );
        }


    }




  })

</script>
</html>
```

前端直接右键 live-server 服务器运行打开即可

后台Controller上加个注解： @CrossOrigin 解决跨域问题

什么是跨域问题？ 百度一下