SpringMVC第三天

第一章：搭建整合环境

1. 搭建整合环境

1. 整合说明：SSM整合可以使用多种方式，咱们会选择XML + 注解的方式
2. 整合的思路
    1. 先搭建整合的环境
    2. 先把Spring的配置搭建完成
    3. 再使用Spring整合SpringMVC框架
    4. 最后使用Spring整合MyBatis框架
3. 创建数据库和表结构
    1. 语句

```
create database ssm;
use ssm;
create table account(
    id int primary key auto_increment,
    name varchar(20),
    money double
);
```

4. 创建maven的工程（今天会使用到工程的聚合和拆分的概念，这个技术maven高级会讲）
    1. 创建ssm_parent父工程（打包方式选择pom，必须的）
    2. 创建ssm_web子模块（打包方式是war包）
    3. 创建ssm_service子模块（打包方式是jar包）
    4. 创建ssm_dao子模块（打包方式是jar包）
    5. 创建ssm_domain子模块（打包方式是jar包）
    6. web依赖于service，service依赖于dao，dao依赖于domain
    7. 在ssm_parent的pom.xml文件中引入坐标依赖

```
<properties>
        <spring.version>5.0.2.RELEASE</spring.version>
        <slf4j.version>1.6.6</slf4j.version>
        <log4j.version>1.2.12</log4j.version>
        <mysql.version>5.1.6</mysql.version>
        <mybatis.version>3.4.5</mybatis.version>
    </properties>

    <dependencies>
```

```
        <!-- spring -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.6.8</version>
        </dependency>

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

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

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>

            <version>4.12</version>
```

```
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.version}</version>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.0</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- log start -->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>${log4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>${slf4j.version}</version>
        </dependency>
        <!-- log end -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>${mybatis.version}</version>

        </dependency>
```

```
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.3.0</version>
        </dependency>

        <dependency>
            <groupId>c3p0</groupId>
            <artifactId>c3p0</artifactId>
            <version>0.9.1.2</version>
            <type>jar</type>
            <scope>compile</scope>
        </dependency>

    </dependencies>

<build>
        <finalName>ssm</finalName>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.2</version>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>
                        <encoding>UTF-8</encoding>
                        <showWarnings>true</showWarnings>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
```

8. 部署ssm_web的项目，只要把ssm_web项目加入到tomcat服务器中即可

5. 编写实体类，在ssm_domain项目中编写

```
package cn.itcast.domain;

import java.io.Serializable;

public class Account implements Serializable{

    private static final long serialVersionUID = 7355810572012650248L;

    private Integer id;
    private String name;
    private Double money;


    public Integer getId() {
```

```
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Double getMoney() {
        return money;
    }
    public void setMoney(Double money) {
        this.money = money;
    }

}
```

6. 编写dao接口

```
package cn.itcast.dao;

import java.util.List;

import cn.itcast.domain.Account;

public interface AccountDao {

    public void saveAccount(Account account);

    public List<Account> findAll();

}
```

7. 编写service接口和实现类

```
package cn.itcast.service;

import java.util.List;

import cn.itcast.domain.Account;

public interface AccountService {
```

```
    public void saveAccount(Account account);

    public List<Account> findAll();

}

package cn.itcast.service.impl;

import java.util.List;

import org.springframework.stereotype.Service;

import cn.itcast.dao.AccountDao;
import cn.itcast.domain.Account;
import cn.itcast.service.AccountService;

@Service("accountService")
public class AccountServiceImpl implements AccountService {

    private AccountDao account;

    public void saveAccount(Account account) {
    }

    public List<Account> findAll() {
        System.out.println("业务层：查询所有账户...");
        return null;
    }

}
```

第二章：Spring框架代码的编写

1. 搭建和测试Spring的开发环境

1. 在ssm_web项目中创建applicationContext.xml的配置文件，编写具体的配置信息。

```
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
```

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

    <!-- 开启注解扫描，要扫描的是service和dao层的注解，要忽略web层注解，因为web层让SpringMVC框架
去管理 -->
    <context:component-scan base-package="cn.itcast">
        <!-- 配置要忽略的注解 -->
        <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

</beans>
```

2. 在ssm_web项目中编写测试方法，进行测试

```
package cn.itcast.test;

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

import cn.itcast.service.AccountService;

public class ServiceTest {

    @Test
    public void run1() {
        ApplicationContext ac = new
ClassPathXmlApplicationContext("classpath:applicationContext.xml");
        AccountService as = (AccountService) ac.getBean("accountService");
        as.findAll();
    }

}
```

第三章：Spring整合SpringMVC框架

1. 搭建和测试SpringMVC的开发环境

1. 在web.xml中配置DispatcherServlet前端控制器

```
    <!-- 配置前端控制器：服务器启动必须加载，需要加载springmvc.xml配置文件 -->
    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- 配置初始化参数，创建完DispatcherServlet对象，加载springmvc.xml配置文件 -->
        <init-param>
```

```
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>
        <!-- 服务器启动的时候，让DispatcherServlet对象创建 -->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

2. 在web.xml中配置DispatcherServlet过滤器解决中文乱码

```
    <!-- 配置解决中文乱码的过滤器 -->
    <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>
    </filter>
    <filter-mapping>
        <filter-name>characterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

3. 创建springmvc.xml的配置文件，编写配置文件

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

    <!-- 扫描controller的注解，别的不扫描 -->
    <context:component-scan base-package="cn.itcast">
        <context:include-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

    <!-- 配置视图解析器 -->
    <bean id="viewResolver"
```

```
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- JSP文件所在的目录 -->
        <property name="prefix" value="/WEB-INF/pages/" />
        <!-- 文件的后缀名 -->
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- 设置静态资源不过滤 -->
    <mvc:resources location="/css/" mapping="/css/**" />
    <mvc:resources location="/images/" mapping="/images/**" />
    <mvc:resources location="/js/" mapping="/js/**" />

    <!-- 开启对SpringMVC注解的支持 -->
    <mvc:annotation-driven />

</beans>
```

释放静态资源的方式二：<mvc:default-servlet-handler/> 此方法只能释放webapp下的静态资源。

4. 测试SpringMVC的框架搭建是否成功
    1. 编写index.jsp和list.jsp编写，超链接

```
<a href="account/findAll">查询所有</a>
```

    2. 创建AccountController类，编写方法，进行测试

```
package cn.itcast.controller;

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

@Controller
@RequestMapping("/account")
public class AccountController {

    /**
     * 查询所有的数据
     * @return
     */
    @RequestMapping("/findAll")
    public String findAll() {
        System.out.println("表现层：查询所有账户...");
        return "list";
    }

}
```

2. Spring整合SpringMVC的框架

1. 目的：在controller中能成功的调用service对象中的方法。

2. 在项目启动的时候，就去加载applicationContext.xml的配置文件，在web.xml中配置ContextLoaderListener监听器（该监听器只能加载WEB-INF目录下的applicationContext.xml的配置文件）。

```
<!-- 配置Spring的监听器 -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>
    <!-- 配置加载类路径的配置文件 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
```

3. 在controller中注入service对象，调用service对象的方法进行测试

```
package cn.itcast.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import cn.itcast.service.AccountService;

@Controller
@RequestMapping("/account")
public class AccountController {

    @Autowired
    private AccountService accoutService;

    /**
     * 查询所有的数据
     * @return
     */
    @RequestMapping("/findAll")
    public String findAll() {
        System.out.println("表现层：查询所有账户...");
        accoutService.findAll();
        return "list";
    }

}
```

第四章：Spring整合MyBatis框架

1. 搭建和测试MyBatis的环境

1. 在web项目中编写SqlMapConfig.xml的配置文件，编写核心配置文件

```
<?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>
    <environments default="mysql">
        <environment id="mysql">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql:///ssm"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>

    <!-- 使用的是注解 -->
    <mappers>
        <!-- <mapper class="cn.itcast.dao.AccountDao"/> -->
        <!-- 该包下所有的dao接口都可以使用 -->
        <package name="cn.itcast.dao"/>
    </mappers>
</configuration>
```

2. 在AccountDao接口的方法上添加注解，编写SQL语句

```
package cn.itcast.dao;

import java.util.List;

import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Select;

import cn.itcast.domain.Account;

public interface AccountDao {

    @Insert(value="insert into account (name,money) values (#{name},#{money})")
    public void saveAccount(Account account);

    @Select("select * from account")
    public List<Account> findAll();

}
```

3. 编写测试的方法

```
package cn.itcast.test;

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

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Test;

import cn.itcast.dao.AccountDao;
import cn.itcast.domain.Account;

public class Demo1 {

    @Test
    public void run1() throws Exception {
        // 加载配置文件
        InputStream inputStream = Resources.getResourceAsStream("SqlMapConfig.xml");
        // 创建工厂
        SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(inputStream);
        // 创建sqlSession对象
        SqlSession session = factory.openSession();
        // 获取代理对象
        AccountDao dao = session.getMapper(AccountDao.class);
        // 调用查询的方法
        List<Account> list = dao.findAll();
        for (Account account : list) {
            System.out.println(account);
        }
        // 释放资源
        session.close();
        inputStream.close();
    }

    @Test
    public void run2() throws Exception {
        Account account = new Account();
        account.setName("熊大");
        account.setMoney(400d);

        // 加载配置文件
        InputStream inputStream = Resources.getResourceAsStream("SqlMapConfig.xml");
        // 创建工厂
        SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(inputStream);
        // 创建sqlSession对象
        SqlSession session = factory.openSession();

        // 获取代理对象
        AccountDao dao = session.getMapper(AccountDao.class);
```

```
        dao.saveAccount(account);

        // 提交事务
        session.commit();
        // 释放资源
        session.close();
        inputStream.close();
    }

}
```

2. Spring整合MyBatis框架

1. 目的：把SqlMapConfig.xml配置文件中的内容配置到applicationContext.xml配置文件中

```
    <!-- 配置C3P0的连接池对象 -->
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql:///ssm" />
        <property name="username" value="root" />
        <property name="password" value="root" />
    </bean>

    <!-- 配置SqlSession的工厂 -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 配置扫描dao的包 -->
    <bean id="mapperScanner" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.itcast.dao"/>
    </bean>
```

id名默认为sqlSessionFactory

如果dao映射配置文件写了别名，可以在这里设置

将扫描下的包变成代理对象，加入到ioc容器中

同时因为与映射文件同包路径下
所以在编译之后他们在同一目录下
这样，在创建代理对象的时候也把配置文件扫描了

2. 在AccountDao接口中添加@Repository注解
3. 在service中注入dao对象，进行测试
4. 代码如下

```
package cn.itcast.dao;

import java.util.List;

import org.apache.ibatis.annotations.Insert;

import org.apache.ibatis.annotations.Select;
```

```
import org.springframework.stereotype.Repository;

import cn.itcast.domain.Account;

@Repository 这个不需要加
public interface AccountDao {

    @Insert(value="insert into account (name,money) values (#{name},#{money})")
    public void saveAccount(Account account);

    @Select("select * from account")
    public List<Account> findAll();

}
```

```
  package cn.itcast.service.impl;

   import java.util.List;

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

   import org.springframework.stereotype.Service;

   import cn.itcast.dao.AccountDao;

   import cn.itcast.domain.Account;

   import cn.itcast.service.AccountService;

   @Service("accountService")

   public class AccountServiceImpl implements AccountService {



    @Autowired

    private AccountDao accountDao;

    public void saveAccount(Account account) {

    }

    public List<Account> findAll() {

        System.out.println("业务层：查询所有账户...");

        return accountDao.findAll();

    }

   }
```

```
package cn.itcast.controller;

import java.util.List;

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

import org.springframework.stereotype.Controller;

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

import cn.itcast.domain.Account;

import cn.itcast.service.AccountService;

@Controller

@RequestMapping("/account")

public class AccountController {



 @Autowired

 private AccountService accoutService;



 /**

  *  查询所有的数据
      * @return
         */
         @RequestMapping("/findAll")
         public String findAll() {
             System.out.println("表现层：查询所有账户...");
             List<Account> list = accoutService.findAll();
             for (Account account : list) {
                 System.out.println(account);
             }
             return "list";
         }

}
```

5. 配置Spring的声明式事务管理

<tx:advice id="txAdvice" transaction-manager="transactionManager">

tx:attributes

<tx:method name="find*" read-only="true"/>

<tx:method name="*" isolation="DEFAULT"/>

/tx:attributes

/tx:advice

aop:config

<aop:advisor advice-ref="txAdvice" pointcut="execution(public * cn.itcast.service..ServiceImpl.*(..))"/>

/aop:config

```
6．测试保存帐户的方法
```

姓名：

金额：

保存

@RequestMapping("/saveAccount") public String saveAccount(Account account) {
accoutService.saveAccount(account); return "list"; }

```

```

补充：

利用maven进行分模块开发时，在web层的配置文件中可以通过<import>标签引入其他模块的配置文件