第14章 SSM框架整合

湖南科技大学
计算机科学与工程学院

学习目标/Target

了解SSM框架的整合思路

熟悉SSM框架整合时的配置文件内容

掌握SSM框架整合应用程序的编写

章节概述/ Summary

对于Java EE应用程序的开发，行业中提供了非常多的技术框架，但是不管如何进行技术选型，Java EE应用都可以分为表现层、业务逻辑层和数据持久层，当前，这3个层的主流框架分别是Spring MVC、Spring和MyBatis，简称为SSM框架，Java EE应用程序也经常通过整合这3大框架来完成开发。SSM框架的整合有多种方式，本章将对图书完稿时常用的整合方式和纯注解的整合方式来对SSM框架的整合进行讲解。

目录/Contents

14.1 常用方式整合SSM框架

14.1.1 整合思路

- 了解常用方式整合思路，能够说出SSM框架整合的大致思路

14.1.1 整合思路

SSM框架整合时三层架构的分工

进行SSM框架整合时，3个框架的分工如下所示。

- MyBatis负责与数据库进行交互。
- Spring负责事务管理，Spring可以管理持久层的Mapper对象及业务层的Service对象。由于Mapper对象和Service对象都在Spring容器中，所以可以在业务逻辑层通过Service对象调用持久层的Mapper对象。
- Spring MVC负责管理表现层的Handler。Spring MVC容器是Spring容器的子容器，因此Spring MVC容器可以调用Spring容器中的Service对象。

14.1.1 整合思路

SSM框架整合实现思路

下面通过一个图书信息查询案例来描述SSM框架的整合，案例实现思路如下。

- 搭建项目基础结构。首先需要在数据库中搭建项目对应的数据库环境；然后创建一个Maven Web项目，并引入案例所需的依赖；最后创建项目的实体类，创建三层架构对应的模块、类和接口。
- 整合Spring和MyBatis。在Spring配置文件中配置数据源信息，并且将SqlSessionFactory对象和Mapper对象都交由Spring管理。
- 整合Spring和Spring MVC。Spring MVC是Spring框架中的一个模块，所以Spring整合Spring MVC只需在项目启动时分别加载各自的配置即可。

14.1.2 项目基础结构搭建

- 熟悉项目基础结构搭建，能够独立完成SSM框架的基础结构搭建

14.1.2 项目基础结构搭建

接下来，根据14.1.1中的整合思路搭建SSM框架整合的项目基础结构，具体如下所示。

STEP 01

搭建数据库环境：MySQL数据库中创建一个名称为ssm的数据库，在该数据库中创建一个名称为tb_book的表，并在tb_book表中插入数据。创建数据库和表，以及往表中插入数据的SQL语句如下所示。

```
CREATE DATABASE ssm;
USE ssm;
CREATE TABLE `tb_book` (
  `id` int(11) ,
  `name` varchar(32) ,
  `press` varchar(32) ,
  `author` varchar(32) );
INSERT INTO `tb_book` VALUES
(1, 'Java EE企业级应用开发教程', '人民邮电出版社', '黑马程序员');
```

14.1.2 项目基础结构搭建

STEP 02

引入项目依赖：本案例中需要引入的相关依赖如下所示。

（1）Spring相关依赖。spring-context：Spring上下文；spring-tx：Spring事务管理；spring-jdbc：SpringJDBC；spring-test：Spring单元测试；spring-webmvc：Spring MVC核心。

（2）MyBatis相关依赖。mybatis：MyBatis核心；

（3）MyBatis与Spring整合包。mybatis-spring ：MyBatis与Spring整合。

（4）数据源相关。druid：阿里提供的数据库连接池。

（5）单元测试相关的依赖。junit：单元测试，与spring-test放在一起做单元测试。

（6）ServletAPI相关的依赖。jsp-api：jsp页面使用request等对象；servlet-api：java文件使用request等对象。

（7）数据库相关的依赖。mysql-connector-java：mysql的数据库驱动包。

14.1.2 项目基础结构搭建

STEP 03

创建实体类：创建名称为Book的实体类。

```
public class Book {
    private Integer id;                 //图书id
    private String name;                //图书名称
    private String press;               //出版社
    private String author;              //作者
    // 省略getter/setter方法
}
```

14.1.2 项目基础结构搭建

STEP 04

创建三层架构对应模块的类和接口：（1）创建名称为BookMapper的持久层接口，在BookMapper接口中定义findBookById()方法，通过图书id获取对应的图书信息。

```
package com.itheima.dao;
import com.itheima.domain.Book;
public interface BookMapper {
    public Book findBookById(Integer id);
}
```

14.1.2 项目基础结构搭建

STEP 04

创建三层架构对应模块的类和接口：（2）创建BookMapper接口对应的映射文件BookMapper.xml。

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.itheima.dao.BookMapper">
    <!--根据id查询图书信息 -->
    <select id="findBookById" parameterType="int"
        resultType="com.itheima.domain.Book">
            select * from book where id = #{id}</select>
</mapper>
```

14.1.2 项目基础结构搭建

STEP 04

创建三层架构对应模块的类和接口：（3）创建名称为BookService的业务层接口，在BookService接口中定义findBookById()方法，通过id获取对应的Book信息。

```
package com.itheima.service;
import com.itheima.domain.Book;
public interface BookService {
    public Book findBookById(Integer id);
}
```

14.1.2 项目基础结构搭建

STEP 04

创建三层架构对应模块的类和接口：（4）创建BookService接口的业务层实现类BookServiceImpl。BookServiceImpl类实现BookService接口的findBookById()方法。

```
@Service
public class BookServiceImpl implements BookService {
    @Autowired
    private BookMapper bookMapper;
    public Book findBookById(Integer id) {
        return bookMapper.findBookById(id);
    }
}
```

14.1.2 项目基础结构搭建

STEP 04

创建三层架构对应模块的类和接口：（5）创建名称为BookController的类。在BookController类中注入一个BookService对象，并且定义一个名称为findBookById()的方法。

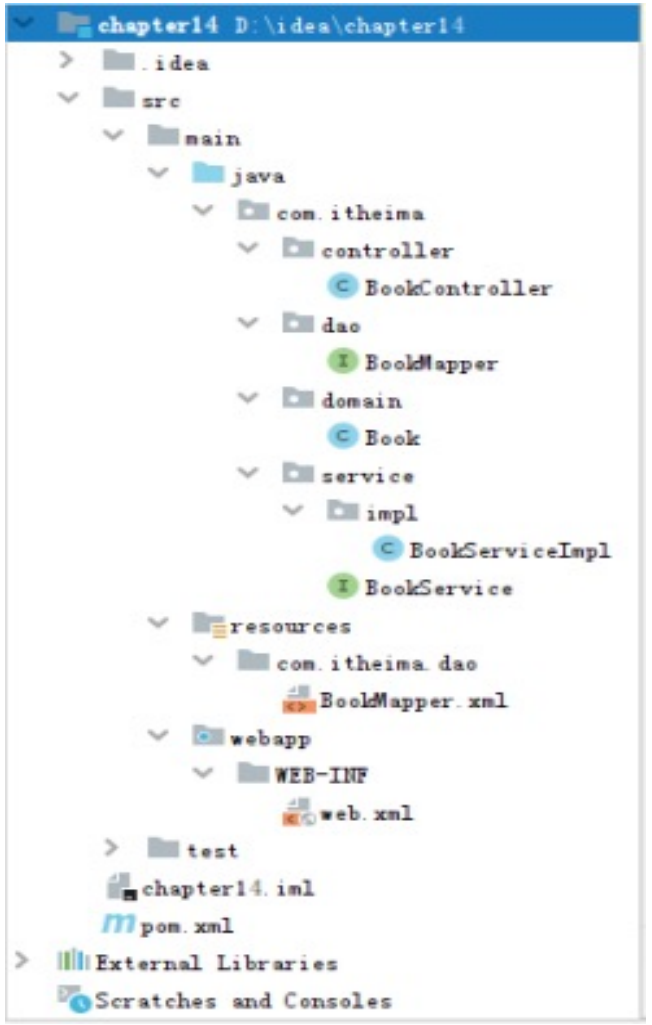
```
@Controller
public class BookController {@Autowired
    private BookService bookService;
    @RequestMapping("/book")
    public ModelAndView findBookById(Integer id){
        Book book = bookService.findBookById(id);
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("book.jsp");
        modelAndView.addObject("book",book); return modelAndView; }}
```

14.1.2 项目基础结构搭建

STEP 04

创建三层架构对应模块的类和接口：

（6）至此，项目基础结构已经搭建完成，项目基础结构如图所示。



14.1.3 Spring和MyBatis整合

- 掌握Spring和Mybatis整合，能够在项目中进行Spring和MyBatis的整合操作

14.1.3 Spring和MyBatis整合

Spring和MyBatis的整合步骤

Spring和MyBatis的整合可以分为2步来完成，首先搭建Spring环境，然后整合MyBatis到Spring环境中。框架环境包含框架对应的依赖和配置文件，其中Spring的依赖、MyBatis的依赖、Spring和MyBatis整合的依赖，在项目基础结构搭建时候已经引入到项目中了，接下来，只需编写Spring的配置文件、Spring和MyBatis整合的配置文件即可。

14.1.3 Spring和MyBatis整合

Spring的配置文件

创建配置文件application-service.xml，用于配置Spring对Service层的扫描信息。application-service.xml具体代码如下所示。

```
<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"
       ...>
    <!--开启注解扫描, 扫描包-->
    <context:component-scan base-package="com.itheima.service"/>
</beans>
```

14.1.3 Spring和MyBatis整合

Spring和MyBatis整合的配置

Spring和MyBatis的整合包中提供了一个SqlSessionFactoryBean对象，该对象的Bean需要注入数据源，也可以根据需求在SqlSessionFactoryBean的Bean中配置MyBatis核心文件路径、别名映射和Mapper映射文件路径。

创建数据源属性文件jdbc.properties，jdbc.properties配置的数据源信息如下所示。

```
jdbc.driverClassName=com.mysql.cj.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssm?useUnicode=true
&characterEncoding=utf-8&serverTimezone=Asia/Shanghai
jdbc.username=root
jdbc.password=root
```

14.1.3 Spring和MyBatis整合

整合测试

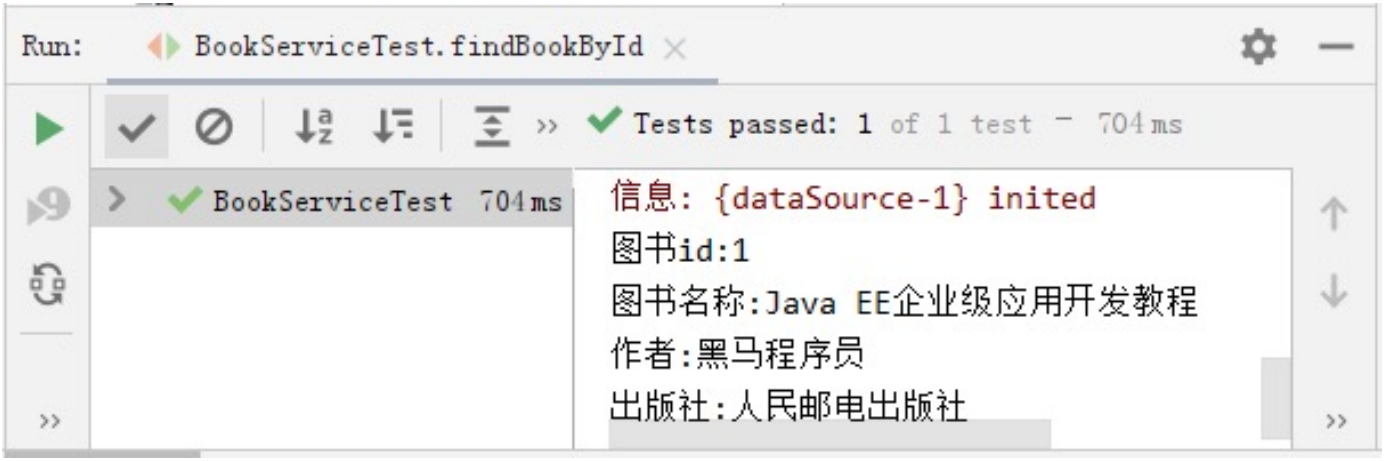
创建名称为BookServiceTest的测试类，用于对Spring和MyBatis的整合进行测试。

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath:application-service.xml",
"classpath:application-dao.xml"})
public class BookServiceTest {
    @Autowired      private BookService bookService;
    @Test
     public void findBookById() {
        Book book = bookService.findBookById(1);
        // 输出语句输出：图书id、图书名称、作者、出版社，省略
}}
```

14.1.3 Spring和MyBatis整合

结果测试

运行测试方法findBookById()，方法运行后控制台打印信息如图所示。



从图中所示的信息可以看出，程序打印出了id为1的图书信息。这表明测试类中成功装配了BookService对象，BookService对象成功调用Service层的findBookById()方法，Service层的findBookById()方法成功调用Dao层的findBookById()方法完成了数据查询。说明Spring和MyBatis已经整合成功。

14.1.4 Spring和Spring MVC整合

- 掌握Spring和SpringMVC整合，能够在项目中进行Spring和Spring MVC的整合操作

14.1.4 Spring和Spring MVC整合

Spring的配置

之前Spring和MyBatis整合时，已经完成了Spring的配置文件，Spring和Spring MVC整合，只需在项目启动时加载Spring容器和Spring的配置文件即可。在项目的web.xml文件中配置Spring的监听器来加载Spring容器及Spring的配置文件，具体配置如下所示。

```
<context-param> <param-name>contextConfigLocation</param-name>
    <param-value>classpath:application-*.xml</param-value>
</context-param>
<listener> <listener-class>
org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

14.1.4 Spring和Spring MVC整合

Spring MVC的配置

本案例主要测试SSM整合的情况，因此在Spring MVC的配置文件中只配置SSM整合案例必须的配置。必须配置的项有以下2个。

- 配置包扫描，指定需要扫描到Spring MVC中的Controller层所在的包路径。
- 配置注解驱动，让项目启动时启用注解驱动，并且自动注册HandlerMapping和HandlerAdapter。

在项目的src\main\resources目录下创建Spring MVC的配置文件spring-mvc.xml。Spring-mvc.xml文件配置完成之后，在web.xml中配置Spring MVC的前端控制器，并在初始化前端控制器时加载Spring MVC的配置文件。

14.1.4 Spring和Spring MVC整合

整合测试

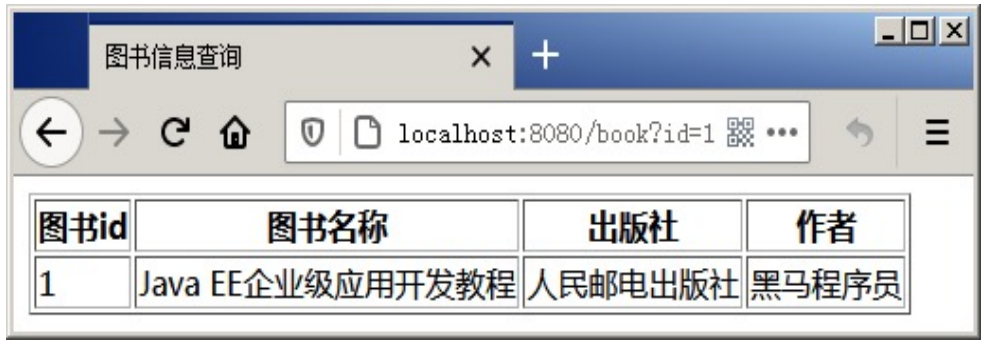
（1）创建名称为book.jsp的文件，用于展示处理器返回的图书信息。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html><head><title>图书信息查询</title></head><body>
<table border="1">
<tr><th>图书id</th><th>图书名称</th>
    <th>出版社</th><th>作者</th></tr>
<tr><td>${book.id}</td><td>${book.name}</td>
    <td>${book.press}</td>
    <td>${book.author}</td></tr>
</table></body>
</html>
```

14.1.4 Spring和Spring MVC整合

整合测试

（2）将chapter14项目部署到Tomcat中，启动项目，在浏览器中访问地址http://localhost:8080/book?id=1来进行图书查询，页面显示效果如图所示。



从图中所示的信息可以看出，程序成功查询到了id为1的图书信息。表明Controller层成功将Service层获取的图书信息返回给页面了，由此可以得出SSM框架整合成功。

14.2 纯注解方式整合SSM框架

14.2.1 整合思路

- 了解整合思路-纯注解方式，能够说出配置类所需要替代的XML配置文件

14.2.1 整合思路

application-dao.xml

application-dao.xml配置文件中配置的内容包含以下4项。

- 读取jdbc.properties文件中的数据连接信息。
- 创建Druid对象，并将读取的数据连接信息注入到Druid数据连接池对象中。
- 创建SqlSessionFactoryBean对象，将并将Druid对象注入到SqlSessionFactoryBean对象中。
- 创建MapperScannerConfigurer对象，并指定扫描的Mapper的路径。

14.2.1 整合思路

application-service.xml和spring-mvc.xml

application-service.xml配置文件中只配置了包扫描，指定需要扫描到Spring 的Service层所在的包路径。spring-mvc.xml配置文件中配置了Spring MVC扫描的包路径和注解驱动。

14.2.1 整合思路

web.xml

web.xml配置文件配置了项目启动时加载的信息，包含如下3个内容。

- 使用<context-param>元素加载Spring配置文件application-service.xml和Spring整合Mybatis的配置文件application-dao.xml。
- Spring容器加载监听器。
- 配置Spring MVC的前端控制器。

14.2.2 纯注解SSM框架整合

- 掌握SSM框架整合-纯注解方式，能够使用纯注解方式整合SSM框架

14.2.2 纯注解SSM框架整合

接下来，将项目中的XML配置文件删除，使用纯注解的配置类依次替换对应的XML文件内容，以完成纯注解的SSM框架整合。具体实现步骤如下所示。

STEP 01

创建名称为JdbcConfig的类，用于获取数据库连接信息并定义创建数据源的对象方法，并定义getDataSource()方法，用于创建DruidDataSource对象。

```
@PropertySource("classpath:jdbc.properties")
public class JdbcConfig {
    // 下面为使用注入的形式。定义dataSource的bean，省略
    @Value("${jdbc.driverClassName}")          private String driver;
    @Value("${jdbc.url}")       private String url;
    @Value("${jdbc.username}")          private String userName;
    @Value("${jdbc.password}")          private String password;
}
```

14.2.2 纯注解SSM框架整合

STEP 02

创建名称为MyBatisConfig的类，在MyBatisConfig类中定义getSqlSessionFactoryBean()方法，用于创建SqlSessionFactoryBean对象并返回。

```
public class MyBatisConfig {
    // 定义MyBatis的核心连接工厂bean
    @Bean
    public SqlSessionFactoryBean getSqlSessionFactoryBean(
                                @Autowired DataSource dataSource){
        SqlSessionFactoryBean ssfb = new SqlSessionFactoryBean();
        ssfb.setDataSource(dataSource); return ssfb;          }
    // 定义MyBatis的映射扫描，省略
}
```

14.2.2 纯注解SSM框架整合

STEP 03

创建名称为SpringConfig的类作为项目定义Bean的源头，并扫描Service层对应的包。

```
@Configuration
@Import({MyBatisConfig.class,JdbcConfig.class})
// 等同于<context:component-scan base-package="com.itheima.service">
@ComponentScan(value = "com.itheima.service")
// 将MyBatisConfig类和JdbcConfig类交给Spring管理
public class SpringConfig {
}
```

14.2.2 纯注解SSM框架整合

STEP 04

创建名称为SpringMvcConfig的类作为Spring MVC的配置类，在配置类中指定Controller层的扫描路径。

```
@Configuration
// 等同于<context:component-scan
// base-package="com.itheima.controller"/>
@ComponentScan("com.itheima.controller")
// 等同于<mvc:annotation-driven/>，还不完全相同
@EnableWebMvc
public class SpringMvcConfig {
}
```

14.2.2 纯注解SSM框架整合

STEP 05

创建名称为ServletContainersInitConfig的类，继承AbstractAnnotationConfigDispatcherServletInitializer抽象类，重写抽象类的方法。用于替代之前web.xml文件配置的信息 ，初始化Servlet容器时加载指定初始化的信息。

```
public class ServletContainersInitConfig extends
            AbstractAnnotationConfigDispatcherServletInitializer {
    // 加载Spring配置类中的信息，初始化Spring容器
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{SpringConfig.class};      }
    // 加载Spring MVC配置类中的信息，初始化Spring MVC容器
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{SpringMvcConfig.class}; }
    // 配置DispatcherServlet的映射路径
    protected String[] getServletMappings() { return new String[]{"/"}; }}
```

14.2.2 纯注解SSM框架整合

AbstractAnnotationConfigDispatcherServletInitializer抽象类

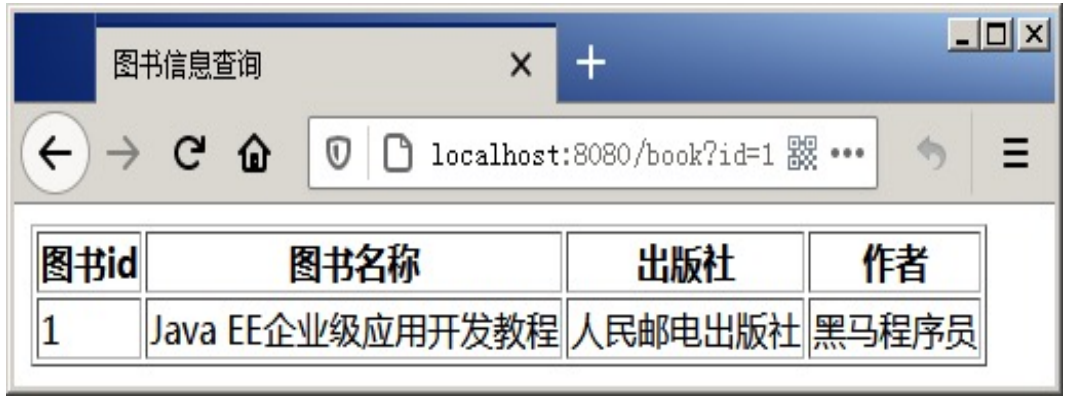
重写AbstractAnnotationConfigDispatcherServletInitializer抽象类的3个方法。

- getRootConfigClasses()方法：将Spring配置类的信息加载到Spring容器中。
- getServletConfigClasses()方法：将Spring MVC配置类的信息加载到Spring MVC容器中。
- getServletMappings()方法：可以指定DispatcherServlet的映射路径。

14.2.2 纯注解SSM框架整合

STEP 06

启动chapter14项目，在浏览器中访问图书信息查询地址，地址为http://localhost:8080/book?id=1，页面显示效果如图所示。



从图中所示的信息可以看出，程序成功查询到了id为1的图书信息。表明Controller将Service获取的图书信息成功返回给页面了，由此可以得出纯注解的SSM框架整合成功。

本章小结

本章主要讲解了SSM框架的整合知识。首先对常用方式整合SSM框架进行了讲解，包括项目基础结构搭建、Spring和MyBatis整合、Spring和Spring MVC整合；然后讲解了纯注解方式整合SSM框架。通过本章的学习，读者将能够了解SSM框架的整合思路，掌握SSM框架的整合过程。SSM框架的整合是SSM框架使用的基础，读者一定要多加练习，并熟练掌握。