



SpringMVC 第三天

第1章 SpringMVC 中的拦截器

1.1 拦截器的作用

Spring MVC 的处理器拦截器类似于 Servlet 开发中的过滤器 Filter，用于对处理器进行预处理和后处理。

用户可以自己定义一些拦截器来实现特定的功能。

谈到拦截器，还要向大家提一个词——拦截器链（Interceptor Chain）。拦截器链就是将拦截器按一定的顺序联结成一条链。在访问被拦截的方法或字段时，拦截器链中的拦截器就会按其之前定义的顺序被调用。

说到这里，可能大家脑海中有了一个疑问，这不是我们之前学的过滤器吗？是的它和过滤器是有几分相似，但是也有区别，接下来我们就来说他们的区别：

过滤器是 servlet 规范中的一部分，任何 java web 工程都可以使用。

拦截器是 SpringMVC 框架自己的，只有使用了 SpringMVC 框架的工程才能用。

过滤器在 url-pattern 中配置了 /* 之后，可以对所有要访问的资源拦截。

拦截器它是只会拦截访问的控制器方法，如果访问的是 jsp, html,css,image 或者 js 是不会进行拦截的。

它也是 AOP 思想的具体应用。

我们要想自定义拦截器， 要求必须实现：**HandlerInterceptor 接口**。

1.2 自定义拦截器的步骤

1.2.1 第一步：编写一个普通类实现 HandlerInterceptor 接口

```
/**
 * 自定义拦截器
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public class HandlerInterceptorDemo1 implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
        throws Exception {
        System.out.println("preHandle 拦截器拦截了");
        return true;
    }
}
```



```
    }

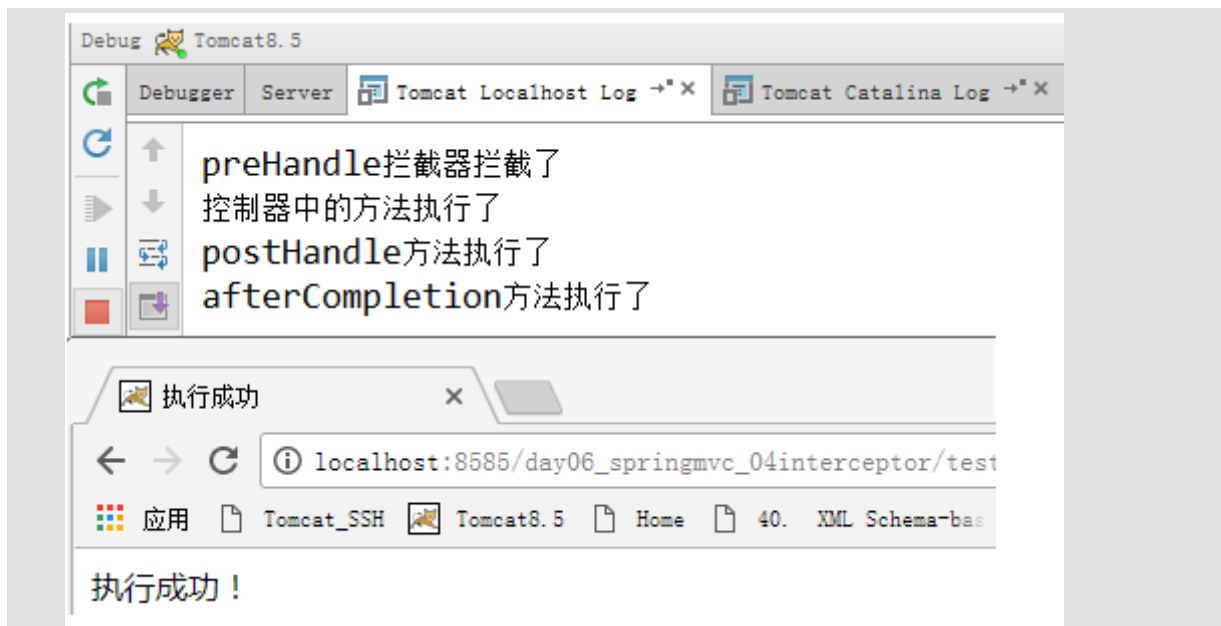
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler,
        ModelAndView modelAndView) throws Exception {
        System.out.println("postHandle 方法执行了");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex)
        throws Exception {
        System.out.println("afterCompletion 方法执行了");
    }
}
```

1.2.2 第二步：配置拦截器

```
<!-- 配置拦截器 -->
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <bean id="handlerInterceptorDemol"
            class="com.itheima.web.interceptor.HandlerInterceptorDemol"></bean>
    </mvc:interceptor>
</mvc:interceptors>
```

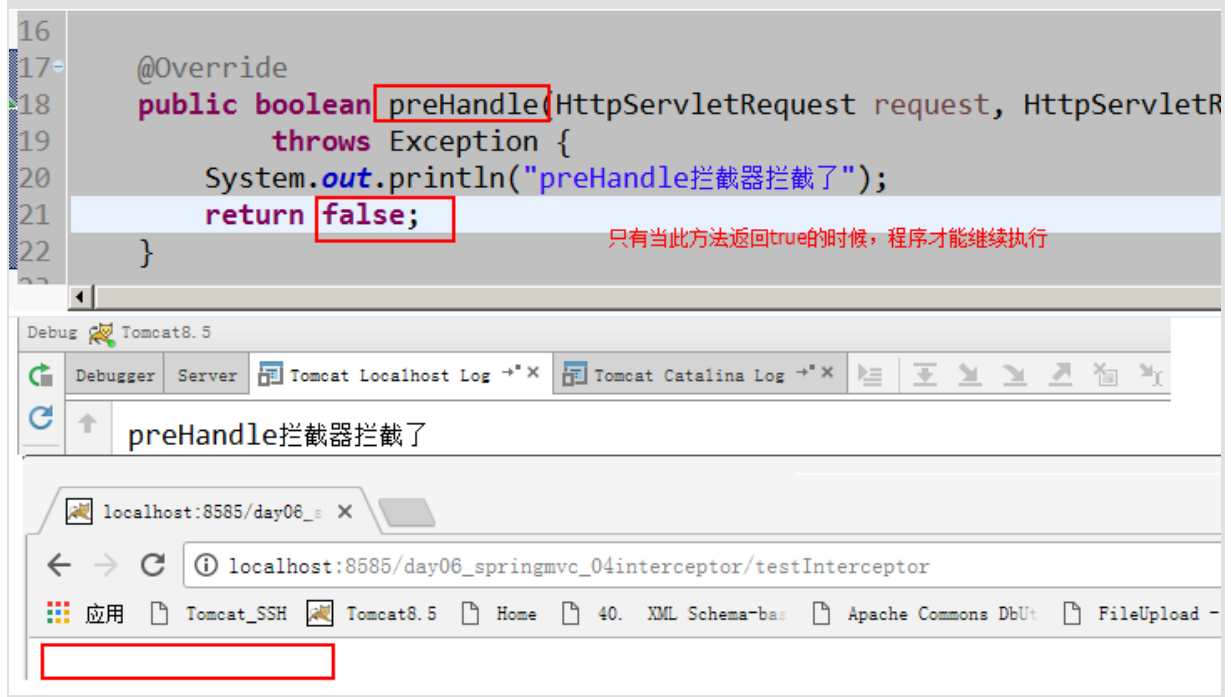
1.2.3 测试运行结果：



1.3 拦截器的细节

1.3.1 拦截器的放行

放行的含义是指，如果有下一个拦截器就执行下一个，如果该拦截器处于拦截器链的最后一个，则执行控制器中的方法。





1.3.2 拦截器中方法的说明

```
public interface HandlerInterceptor {

    /**
     * 如何调用：
     *      按拦截器定义顺序调用
     * 何时调用：
     *      只要配置了都会调用
     * 有什么用：
     *      如果程序员决定该拦截器对请求进行拦截处理后还要调用其他的拦截器，或者是业务处理器去
     *      进行处理，则返回 true。
     *      如果程序员决定不需要再调用其他的组件去处理请求，则返回 false。
     */
    default boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler)
        throws Exception {
        return true;
    }

    /**
     * 如何调用：
     *      按拦截器定义逆序调用
     * 何时调用：
     *      在拦截器链内所有拦截器返回成功调用
     * 有什么用：
     *      在业务处理器处理完请求后，但是 DispatcherServlet 向客户端返回响应前被调用，
     *      在该方法中对用户请求 request 进行处理。
     */
    default void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler,
        @Nullable ModelAndView modelAndView) throws Exception {
    }

    /**
     * 如何调用：
     *      按拦截器定义逆序调用
     * 何时调用：
     *      只有 preHandle 返回 true 才调用
     * 有什么用：
     *      在 DispatcherServlet 完全处理完请求后被调用，
     *      可以在该方法中进行一些资源清理的操作。
     */
    default void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler,
```



```
        @Nullable Exception ex) throws Exception {  
  
    }  
}
```

思考:

如果有多个拦截器，这时拦截器 1 的 preHandle 方法返回 true，但是拦截器 2 的 preHandle 方法返回 false，而此时拦截器 1 的 afterCompletion 方法是否执行？

1.3.3 拦截器的作用路径

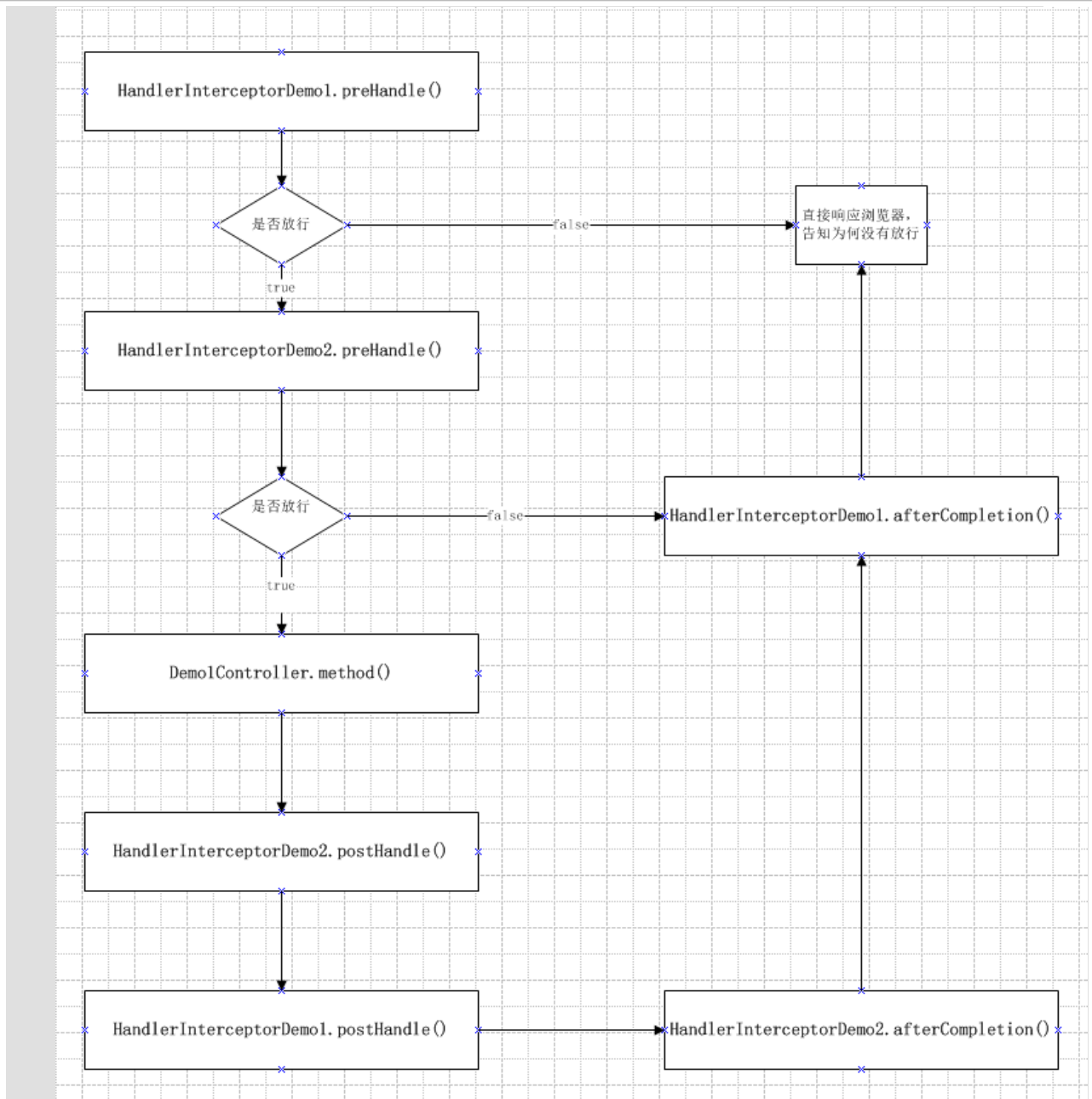
作用路径可以通过在配置文件中配置。

<!-- 配置拦截器的作用范围 -->

```
<mvc:interceptors>  
    <mvc:interceptor>  
        <mvc:mapping path="/**" /><!-- 用于指定对拦截的 url -->  
        <mvc:exclude-mapping path="/" /><!-- 用于指定排除的 url -->  
        <bean id="handlerInterceptorDemo1"  
            class="com.ithema.web.interceptor.HandlerInterceptorDemo1"></bean>  
    </mvc:interceptor>  
</mvc:interceptors>
```

1.3.4 多个拦截器的执行顺序

多个拦截器是按照配置的顺序决定的。



1.4 正常流程测试

1.4.1 配置文件:

```

<!-- 配置拦截器的作用范围 -->
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/*" /><!-- 用于指定对拦截的 url -->
    <bean id="handlerInterceptorDemo1"
class="com.itheima.web.interceptor.HandlerInterceptorDemo1"></bean>
  </mvc:interceptor>

```



```
<mvc:interceptor>
    <mvc:mapping path="/*" />
    <bean id="handlerInterceptorDemo2"
class="com.ithiema.web.interceptor.HandlerInterceptorDemo2"></bean>
    </mvc:interceptor>
</mvc:interceptors>
```

1.4.2 拦截器 1 的代码：

```
/**
 * 自定义拦截器
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public class HandlerInterceptorDemo1 implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler)
        throws Exception {
        System.out.println("拦截器 1: preHandle 拦截器拦截了");
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler,
        ModelAndView modelAndView) throws Exception {
        System.out.println("拦截器 1: postHandle 方法执行了");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex)
        throws Exception {
        System.out.println("拦截器 1: afterCompletion 方法执行了");
    }
}
```



1.4.3 拦截器 2 的代码：

```
/**
 * 自定义拦截器
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public class HandlerInterceptorDemo2 implements HandlerInterceptor {

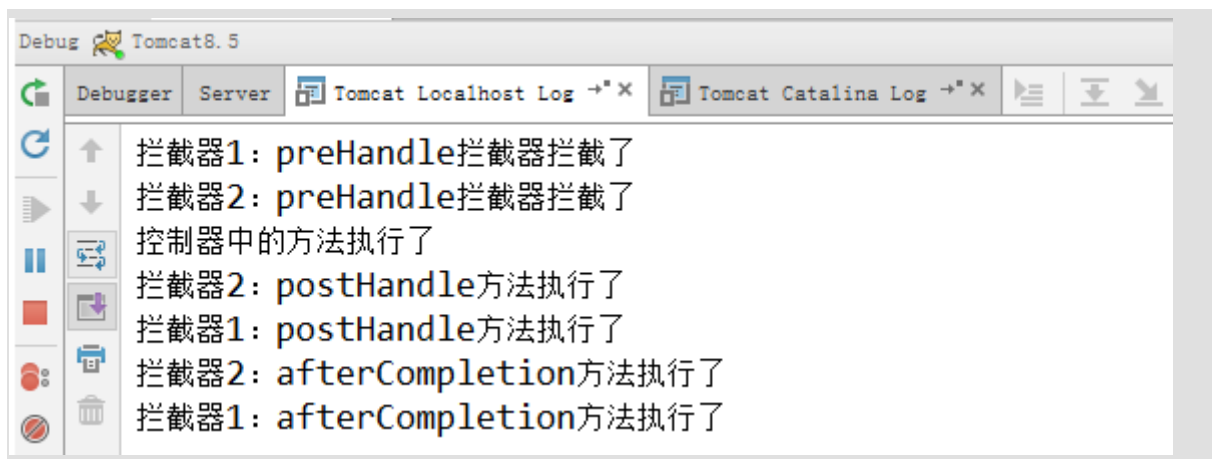
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler)
        throws Exception {
        System.out.println("拦截器 2: preHandle 拦截器拦截了");
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler,
        ModelAndView modelAndView) throws Exception {
        System.out.println("拦截器 2: postHandle 方法执行了");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex)
        throws Exception {
        System.out.println("拦截器 2: afterCompletion 方法执行了");
    }
}
```




1.4.4 运行结果：



1.5 中断流程测试

1.5.1 配置文件：

```
<!-- 配置拦截器的作用范围 -->
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/**" /><!-- 用于指定对拦截的 url -->
        <bean id="handlerInterceptorDemo1"
            class="com.itheima.web.interceptor.HandlerInterceptorDemo1"></bean>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**" />
        <bean id="handlerInterceptorDemo2"
            class="com.itheima.web.interceptor.HandlerInterceptorDemo2"></bean>
    </mvc:interceptor>
</mvc:interceptors>
```

1.5.2 拦截器 1 的代码：

```
/**
 * 自定义拦截器
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public class HandlerInterceptorDemo1 implements HandlerInterceptor {
```



```
@Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler)
        throws Exception {
        System.out.println("拦截器 1: preHandle 拦截器拦截了");
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler,
        ModelAndView modelAndView) throws Exception {
        System.out.println("拦截器 1: postHandle 方法执行了");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex)
        throws Exception {
        System.out.println("拦截器 1: afterCompletion 方法执行了");
    }
}
```

1.5.3 拦截器 2 的代码：

```
/**
 * 自定义拦截器
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public class HandlerInterceptorDemo2 implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler)
        throws Exception {
        System.out.println("拦截器 2: preHandle 拦截器拦截了");
        return false;
    }

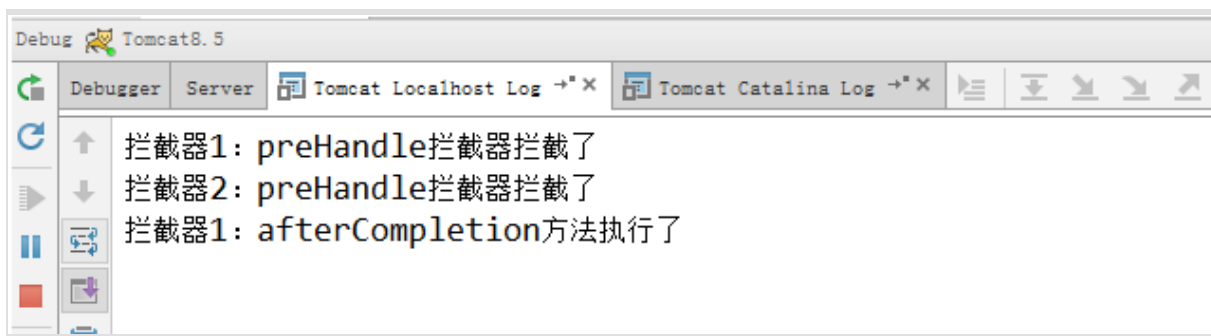
    @Override
```



```
public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler,
    ModelAndView modelAndView) throws Exception {
    System.out.println("拦截器 2: postHandle 方法执行了");
}

@Override
public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex)
    throws Exception {
    System.out.println("拦截器 2: afterCompletion 方法执行了");
}
}
```

1.5.4 运行结果：



1.6 拦截器的简单案例（验证用户是否登录）

1.6.1 实现思路

- 1、有一个登录页面，需要写一个 controller 访问页面
- 2、登录页面有一提交表单的动作。需要在 controller 中处理。
 - 2.1、判断用户名密码是否正确
 - 2.2、如果正确 向 session 中写入用户信息
 - 2.3、返回登录成功。
- 3、拦截用户请求，判断用户是否登录
 - 3.1、如果用户已经登录。放行
 - 3.2、如果用户未登录，跳转到登录页面

1.6.2 控制器代码

```
// 登陆页面
@RequestMapping("/login")
```



```
public String login(Model model) throws Exception{
    return "login";
}

//登陆提交
//userid: 用户账号, pwd: 密码
@RequestMapping("/loginsubmit")
public String loginsubmit(HttpSession session,String userid,String pwd) throws
Exception{

    //向 session 记录用户身份信息
    session.setAttribute("activeUser", userid);
    return "redirect:/main.jsp";
}

//退出
@RequestMapping("/logout")
public String logout(HttpSession session) throws Exception{

    //session 过期
    session.invalidate();

    return "redirect:index.jsp";
}
```

1.6.3 拦截器代码

```
public class LoginInterceptor implements HandlerInterceptor{

    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {

        //如果是登录页面则放行
        if(request.getRequestURI().indexOf("login.action")>=0){
            return true;
        }
        HttpSession session = request.getSession();
        //如果用户已登录也放行
        if(session.getAttribute("user")!=null){
            return true;
        }
        //用户没有登录挑战到登录页面
        request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request,
```

```
response);  
  
    return false;  
  
}  
  
}
```

第2章 SSM 整合

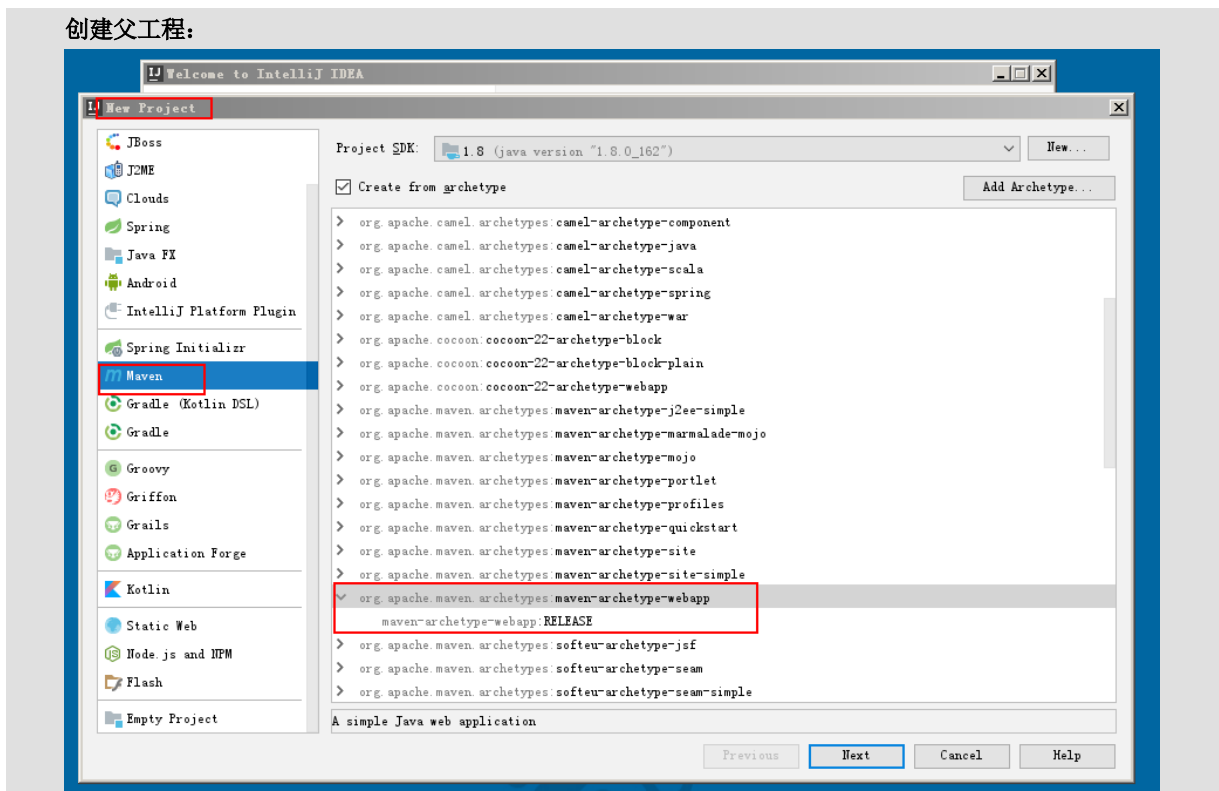
2.1 环境准备

2.1.1 创建数据库和表结构

```
create database ssm;  
create table account(  
    id int primary key auto_increment,  
    name varchar(100),  
    money double(7,2),  
);
```

2.1.2 创建 Maven 工程

创建父工程:





2.1.3 导入坐标并建立依赖

注意 MyBatis 和 Spring 的版本对应关系：

MyBatis-Spring	MyBatis	Spring
1.0.0 and 1.0.1	3.0.1 to 3.0.5	3.0.0 or higher
1.0.2	3.0.6	3.0.0 or higher
1.1.0 or higher	3.1.0 or higher	3.0.0 or higher
1.3.0 or higher	3.4.0 or higher	3.0.0 or higher

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.7</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.0</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
  </dependency>
</dependencies>
```



```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.5</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.1</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>jstl</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
</dependencies>
```

2.1.4 编写实体类

```
/**
 * 账户的实体类
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public class Account implements Serializable {

    private Integer id;
    private String name;
```



```
private Float 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 Float getMoney() {
    return money;
}
public void setMoney(Float money) {
    this.money = money;
}
@Override
public String toString() {
    return "Account [id=" + id + ", name=" + name + ", money=" + money + "];"
}
}
```

2.1.5 编写业务层接口

```
/**
 * 账户的业务层接口
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public interface IAccountService {

    /**
     * 保存账户
     * @param account
     */
    void saveAccount(Account account);

    /**
     * 查询所有账户
     */
}
```




```
    * @return  
    */  
    List<Account> findAllAccount();  
  
}
```

2.1.6 编写持久层接口

```
/**  
 * 账户的持久层接口  
 * @author 黑马程序员  
 * @Company http://www.ithiema.com  
 * @Version 1.0  
 */  
public interface IAccountDao {  
  
    /**  
     * 保存  
     * @param account  
     */  
    void save(Account account);  
  
    /**  
     * 查询所有  
     * @return  
     */  
    List<Account> findAll();  
  
}
```

2.2 整合步骤

2.2.1 保证 Spring 框架在 web 工程中独立运行

2.2.1.1 第一步：编写 spring 配置文件并导入约束

```
<?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:aop="http://www.springframework.org/schema/aop"
        xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/tx
            http://www.springframework.org/schema/tx/spring-tx.xsd
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context.xsd">
<!-- 配置 spring 创建容器时要扫描的包 -->
<context:component-scan base-package="com.itheima">
    <!--制定扫包规则，不扫描@Controller 注解的 JAVA 类，其他的还是要扫描 -->
    <context:exclude-filter type="annotation"
        expression="org.springframework.stereotype.Controller" />
</context:component-scan>
</beans>

```

2.2.1.2 第二步：使用注解配置业务层和持久层

```

/**
 * 账户的业务层实现类
 */
@Service("accountService")
public class AccountServiceImpl implements IAccountService {

    @Autowired
    private IAccountDao accountDao;

    @Override
    public List<Account> findAllAccount() {
        return accountDao.findAllAccount();
    }

    @Override
    public void saveAccount(Account account) {
        accountDao.saveAccount
    }
}

```

持久层实现类代码：

此时不要做任何操作，就输出一句话。目的是测试 spring 框架搭建的结果。



```
/**
 * 账户的持久层实现类
 */
@Repository("accountDao")
public class AccountDaoImpl implements IAccountDao {

    @Override
    public List<Account> findAllAccount() {
        System.out.println("查询了所有账户");
        return null;
    }

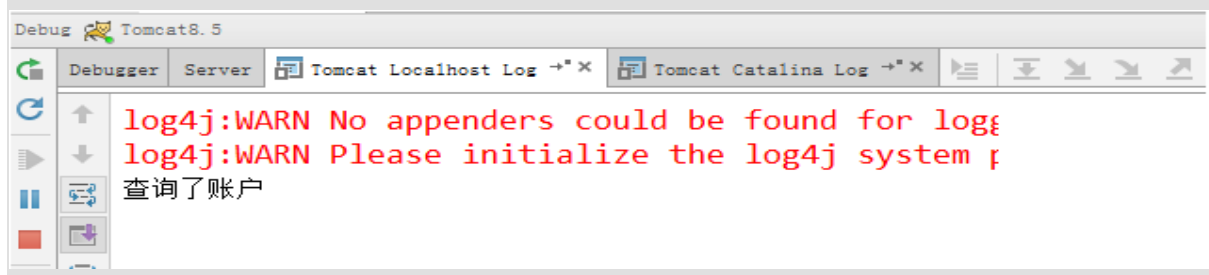
    @Override
    public void saveAccount(Account account) {
        System.out.println("保存了账户");
    }
}
```

2.2.1.3 第三步：测试 spring 能否独立运行

```
/**
 * 测试 spring 环境搭建是否成功
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public class Test01Spring {

    public static void main(String[] args) {
        ApplicationContext ac = new
        ClassPathXmlApplicationContext("applicationContext.xml");
        IAccountService as = ac.getBean("accountService", IAccountService.class);
        as.findAllAccount();
    }
}
```

运行结果：





2.2.2 保证 SpringMVC 在 web 工程中独立运行

2.2.2.1 第一步：在 web.xml 中配置核心控制器（DispatcherServlet）

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">
    <display-name>ssm_web</display-name>

    <!-- 配置 spring mvc 的核心控制器 -->
    <servlet>
        <servlet-name>springmvcDispatcherServlet</servlet-name>

        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- 配置初始化参数，用于读取 springmvc 的配置文件 -->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>
        <!-- 配置 servlet 的对象的创建时间点：应用加载时创建。取值只能是非 0 正整数，表示启动顺
序 -->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvcDispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- 配置 springMVC 编码过滤器 -->
    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>

        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-cla
ss>

        <!-- 设置过滤器中的属性值 -->
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <!-- 启动过滤器 -->
```



```
<init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
</init-param>
</filter>
<!-- 过滤所有请求 -->
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

2.2.2.2 第二步：编写 SpringMVC 的配置文件

```
<?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">

    <!-- 配置创建 spring 容器要扫描的包 -->
    <context:component-scan base-package="com.itheima">
        <!-- 制定扫描规则，只扫描使用@Controller 注解的 JAVA 类 -->
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Controller" />
    </context:component-scan>

    <!-- 配置视图解析器 -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
```



```
<property name="prefix" value="/WEB-INF/pages/"></property>
<property name="suffix" value=".jsp"></property>
</bean>

<mvc:annotation-driven></mvc:annotation-driven>
</beans>
```

2.2.2.3 第三步：编写 Controller 和 jsp 页面

jsp 代码：

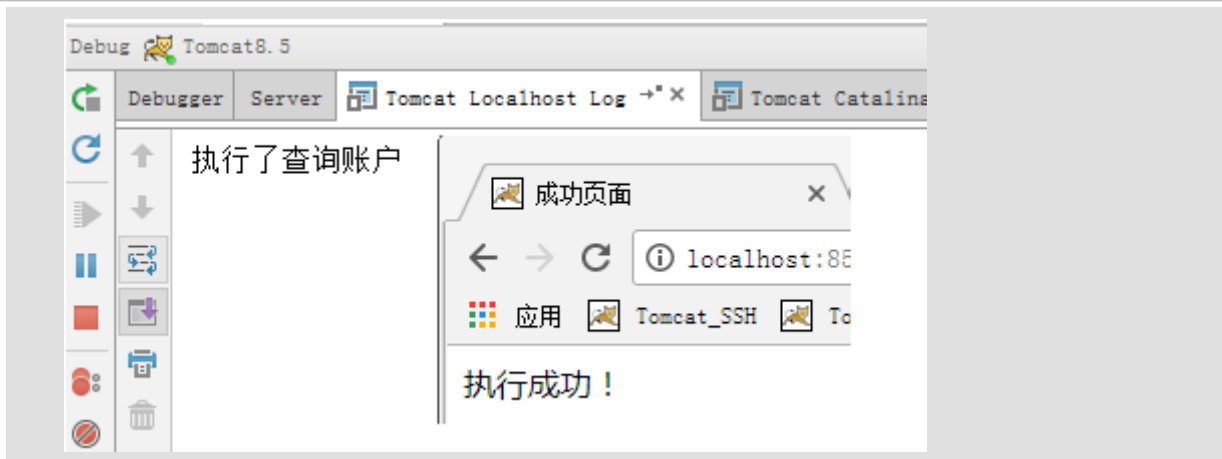
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>主页</title>
</head>
<body>
<a href="account/findAllAccount">访问查询账户</a>
</body>
</html>
```

控制器代码：

```
/**
 * 账户的控制器
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
@Controller("accountController")
@RequestMapping("/account")
public class AccountController {

    @RequestMapping("/findAllAccount")
    public String findAllAccount() {
        System.out.println("执行了查询账户");
        return "success";
    }
}
```

运行结果：



2.2.3 整合 Spring 和 SpringMVC

2.2.3.1 第一步：配置监听器实现启动服务创建容器

```
<!-- 配置 spring 提供的监听器，用于启动服务时加载容器 。
      该监听器只能加载 WEB-INF 目录中名称为 applicationContext.xml 的配置文件 -->
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>

<!-- 手动指定 spring 配置文件位置 -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
</context-param>
```

2.2.4 保证 MyBatis 框架在 web 工程中独立运行

2.2.4.1 第一步：编写 AccountDao 映射配置文件

注意：我们使用代理 dao 的方式来操作持久层，所以此处 Dao 的实现类就是多余的了。

```
<?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.IAccountDao">
```



```
<!-- 查询所有账户 -->
<select id="findAll" resultType="com.itheima.domain.Account">
    select * from account
</select>

<!-- 新增账户 -->
<insert id="save" parameterType="com.itheima.domain.Account">
    insert into account(name,money) values(#{name},#{money});
</insert>
</mapper>
```

2.2.4.2 第二步：编写 SqlMapConfig 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>
    <properties resource="jdbcConfig.properties"></properties>
    <environments default="mysql">
        <environment id="mysql">
            <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>
    <mappers>
        <mapper resource="com/itheima/dao/AccountDao.xml"/>
    </mappers>
</configuration>
```

properties 文件中的内容：

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssm
jdbc.username=root
jdbc.password=1234
```




2.2.4.3 第三步：测试运行结果

测试类代码：

```
/**
 * 测试 MyBatis 独立使用
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
public class Test02MyBatis {

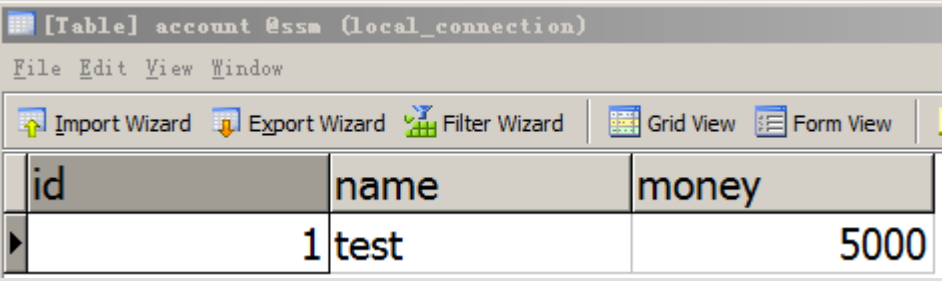
    /**
     * 测试保存
     * @param args
     * @throws Exception
     */
    @Test
    public void testSave() throws Exception {
        Account account = new Account();
        account.setName("test");
        account.setMoney(5000f);
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");
        SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);
        SqlSession session= factory.openSession();

        IAccountDao aDao = session.getMapper(IAccountDao.class);
        aDao.save(account);
        session.commit();
        session.close();
        in.close();
    }

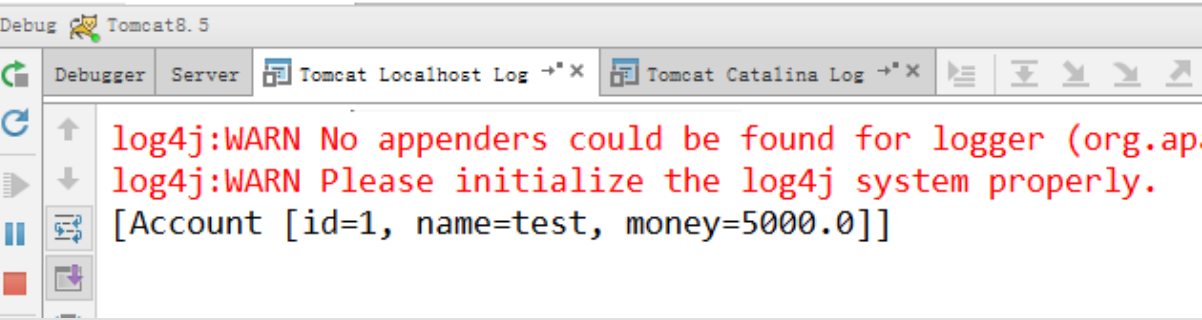
    /**
     * 测试查询
     * @param args
     * @throws Exception
     */
    @Test
    public void testFindAll() throws Exception{
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");
        SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);
        SqlSession session= factory.openSession();
        IAccountDao aDao = session.getMapper(IAccountDao.class);
    }
}
```



```
List<Account> list = aDao.findAll();
System.out.println(list);
session.close();
in.close();
}
}
```



id	name	money
1	test	5000



2.2.5 整合 Spring 和 MyBatis

整合思路:

把 mybatis 配置文件 (SqlMapConfig.xml) 中内容配置到 spring 配置文件中
同时, 把 mybatis 配置文件的内容清掉。

```
<?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>
</configuration>
```

注意:

由于我们使用的是代理 Dao 的模式, Dao 具体实现类由 MyBatis 使用代理方式创建, 所以此时 mybatis 配置文件不能删。

当我们整合 spring 和 mybatis 时, mybatis 创建的 Mapper.xml 文件名必须和 Dao 接口文件名一致

2.2.5.1 第一步: Spring 接管 MyBatis 的 Session 工厂

```
<!-- 加载配置文件 -->
<context:property-placeholder location="classpath:jdbcConfig.properties" />
```



```
<!-- 配置 MyBatis 的 Session 工厂 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <!-- 数据库连接池 -->
    <property name="dataSource" ref="dataSource" />
    <!-- 加载 mybatis 的全局配置文件 -->
    <property name="configLocation" value="classpath:SqlMapConfig.xml" />
</bean>

<!-- 配置数据源 -->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="${jdbc.driver}"></property>
    <property name="jdbcUrl" value="${jdbc.url}"></property>
    <property name="user" value="${jdbc.username}"></property>
    <property name="password" value="${jdbc.password}"></property>
</bean>
```

2.2.5.2 第二步：配置自动扫描所有 Mapper 接口和文件

```
<!-- 配置 Mapper 扫描器 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.itheima.dao"/>
</bean>
```

2.2.5.3 第三步：配置 spring 的事务

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

<!-- 配置事务的通知 -->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*" propagation="REQUIRED" read-only="false"/>
        <tx:method name="find*" propagation="SUPPORTS" read-only="true"/>
    </tx:attributes>
</tx:advice>

<!-- 配置 aop -->
<aop:config>
<!-- 配置切入点表达式 -->
```



```
<aop:pointcut expression="execution(* com.ithiema.service.impl.*.*(..))"
id="pt1"/>
<!-- 建立通知和切入点表达式的关系 -->
<aop:advisor advice-ref="txAdvice" pointcut-ref="pt1"/>
</aop:config>
```

2.2.5.4 第三步：测试整合结果

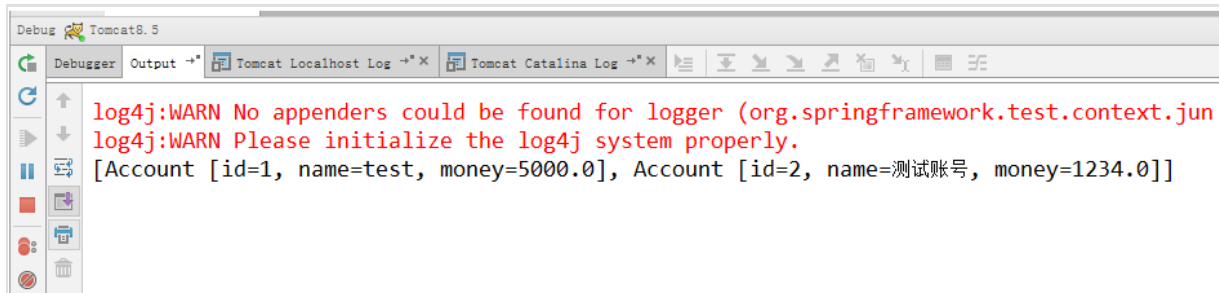
```
/**
 * 测试 spring 整合 mybatis
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations= {"classpath:applicationContext.xml"})
public class Test03SpringMabatis {

    @Autowired
    private IAccountService accountService;

    @Test
    public void testFindAll() {
        List list = accountService.findAllAccount();
        System.out.println(list);
    }

    @Test
    public void testSave() {
        Account account = new Account();
        account.setName("测试账号");
        account.setMoney(1234f);
        accountService.saveAccount(account);
    }
}
```

[Table] account @ssm (local_connection)		
File Edit View Window		
Import Wizard Export Wizard Filter Wizard Grid View Form View		
id	name	money
1	test	5000
2	测试账号	1234



2.2.6 测试 SSM 整合结果

2.2.6.1 编写测试 jsp

请求发起页面:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>主页</title>
</head>
<body>
<a href="account/findAllAccount">访问查询账户</a>
<hr/>
<form action="account/saveAccount" method="post">
    账户名称: <input type="text" name="name"/><br/>
    账户金额: <input type="text" name="money"/><br/>
    <input type="submit" value="保存"/>
</form>
</body>
</html>
```

响应结果页面:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>账户的列表页面</title>
```



```
</head>
<body>
    <table border="1" width="300px">
        <tr>
            <th>编号</th>
            <th>账户名称</th>
            <th>账户金额</th>
        </tr>
        <c:forEach items="${accounts}" var="account" varStatus="vs">
            <tr>
                <td>${vs.count}</td>
                <td>${account.name }</td>
                <td>${account.money }</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

2.2.6.2 修改控制器中的方法

```
/**
 * 账户的控制器
 * @author 黑马程序员
 * @Company http://www.ithiema.com
 * @Version 1.0
 */
@Controller("accountController")
@RequestMapping("/account")
public class AccountController {

    @Autowired
    private IAccountService accountService;

    /**
     * 查询所有账户
     * @return
     */
    @RequestMapping("/findAllAccount")
    public ModelAndView findAllAccount() {
        List<Account> accounts = accountService.findAllAccount();
        ModelAndView mv = new ModelAndView();
        mv.addObject("accounts", accounts);
        mv.setViewName("accountlist");
    }
}
```



```
        return mv;
    }

    /**
     * 保存账户
     * @param account
     * @return
     */
    @RequestMapping("/saveAccount")
    public String saveAccount(Account account) {
        accountService.saveAccount(account);
        return "redirect:findAllAccount";
    }
}
```

2.2.6.3 测试运行结果

The screenshot displays a web application interface for account management. The top left window shows a form titled "访问查询账户" (Access Query Account) with input fields for "账户名称" (Account Name) set to "泰斯特" (Taiste) and "账户金额" (Account Amount) set to "20000", and a "保存" (Save) button. The top right window shows a table of accounts with columns "编号" (ID), "账户名称" (Account Name), and "账户金额" (Account Amount). The bottom window shows a detailed view of the account table with columns "id", "name", and "money".

编号	账户名称	账户金额
1	test	5000.0
2	测试账号	1234.0
3	泰斯特	20000.0

id	name	money
1	test	5000
2	测试账号	1234
3	泰斯特	20000