springMVC的第三天

一、教学目标

1. 整合ssm框架
2. 拦截器

二、整合SSM

1、整合思路

```
a. SSM介绍
    mybatis 持久层的CURD
    spring 业务层  IOC、DI 和AOP
    springMVC 表现层 MVC的操作
b. 整合使用的技术
    1). Spring      5.0.2
    2). mybatis     3.4.5
    3). SpringMVC   5.0.2
    4). log4J2      2.9.1
    5). bootstrap   3.3.5
    6). jquery      1.9.1
    ......
c. 业务介绍
    我们需要完成一张账户表的增删改查操作
```

2、引入依赖和依赖分析

```
a. Spring相关的
    1). spring-context : Spring容器
    2). spring-tx : Spring事务
    3). spring-jdbc : SpringJDBC
    4). spring-test : Spring单元测试
    5). spring-webmvc : SpringMVC
b. mybatis相关的
    1). mybatis : mybatis核心
    2). mybatis-spring : mybatis与spring整合
    3) 切面相关的
        aspectjweaver : AOP切面
    4) 数据源相关（选择使用）:
        c3p0
        commons-dbcp
        spring自带的数据源
    5) 单元测试相关的:
        junit : 单元测试,与spring-test放一起做单元测试
    6) ServletAPI相关的
        jsp-api : jsp页面使用request等对象

        servlet-api : java文件使用request等对象
```

```
    7) 日志相关的 :
        log4j-core : log4j2核心包
        log4j-api : log4j2的功能包
        log4j-web : web项目相关日志功能
        slf4j-api : 另外一种日志包,slf4j:Simple Logging Facade for Java
        为java做简单的日志记录此处和log4j一起
        log4j-slf4j-impl : slf4j的log4j实现类,也就是说slf4j的日志记录功能由log4j实现
        log4j-jcl : 程序运行的时候检测用了哪种日志实现类现在叫Apache Common Logging
    8) 数据库相关的
        mysql-connector-java : mysql的数据库驱动包
    9) 页面表达式
        JSTL : JSTL标签库必须jar包 基础功能
        standard : JSTL标签库的必须jar包 进阶功能
    10) 文件上传
        commons-fileupload : 上传插件
        commons-io : IO操作包
```

3、表和实体类的创建

```
a. mysql创建表
    CREATE TABLE `account` (
      `id` int(11) NOT NULL AUTO_INCREMENT,
      `username` varchar(20) DEFAULT NULL,
      `money` float(8,2) DEFAULT NULL,
      PRIMARY KEY (`id`)
    ) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8;
b. 实体类创建
    public class Account {

        private Integer id;
        private String username;
        private Double money;

        // 此处省略set和get方法
    }
```

4、Dao层的编写

```
package com.itheima.dao;

import com.itheima.domain.Account;
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Update;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.stereotype.Repository;

import java.util.List;

/**
 * 账户的持久层
```

```
 */
@Repository
@MapperScan
public interface AccountDao {
    /**
     * 查全部
     * @return
     */
    @Select("select * from account")
    public List<Account> findAll();

    /**
     * 根据id查询
     * @param id
     * @return
     */
    @Select("select * from account where id = #{id}")
    public Account findById(Integer id);

    /**
     * 保存账户
     * @param account
     */
    @Insert("insert into account values(null,#{username},#{money})")
    public void saveAccount(Account account);

    /**
     * 更新账户
     * @param account
     */
    @Update("update account set username = #{username} , money = #{money} where id = #{id}")
    public void updateAccount(Account account);

    /**
     * 根据id删除账户
     * @param id
     */
    @Delete("delete from account where id = #{id}")
    public void deleteById(Integer id);

    /**
     * 根据用户名查询账户
     * @param username
     * @return
     */
    @Select("select * from account where username = #{username}")
    public Account findByUsername(String username);
}
```

5、Service层接口的编写

```
package com.itheima.service;
```

```
import com.itheima.domain.Account;

import java.util.List;

public interface AccountService {

    /**
     * 查全部
     * @return
     */
    public List<Account> findAll();

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

    /**
     * 更新账户
     * @param account
     */
    public void updateAccount(Account account);

    /**
     * 根据id删除账户
     * @param id
     */
    public void deleteById(Integer id);

    /**
     * 转账
     * @param fromUsername
     * @param toUsername
     * @param money
     */
    public void transfer(String fromUsername,String toUsername ,Double money) ;

    /**
     * 根据id查询
     * @param id
     */
    Account updateAccountUI(Integer id);
}
```

6、Service层实现类编写

```
package com.itheima.service.impl;

import com.itheima.dao.AccountDao;

import com.itheima.domain.Account;
```

```java
import com.itheima.service.AccountService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * 账户的业务层
 */
@Service
public class AccountServiceImpl implements AccountService {

    @Autowired
    AccountDao accountDao;

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

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

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

    @Override
    public void deleteById(Integer id) {
        accountDao.deleteById(id);
    }

    public void transfer(String fromUsername,String toUsername ,Double money) {
        Account fromAccount = accountDao.findByUsername(fromUsername);

        Account toAccount = accountDao.findByUsername(toUsername);

        fromAccount.setMoney(fromAccount.getMoney() - money);
        toAccount.setMoney(toAccount.getMoney() + money);

        accountDao.updateAccount(fromAccount);

        accountDao.updateAccount(toAccount);
    }

    @Override
    public Account updateAccountUI(Integer id) {
        return accountDao.findById(id);
    }

}
```

6、spring和mybatis的整合

```
1. SqlMapConfig的配置
    null
2. applicationContext的配置
    <!--1. 引入属性配置文件-->
    <context:property-placeholder location="classpath:db.properties"></context:property-
placeholder>
    <!--2. 配置数据源，代替mybatis中的数据源配置-->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${jdbc.driver}"></property>
        <property name="url" value="${jdbc.url}"></property>
        <property name="username" value="${jdbc.user}"></property>
        <property name="password" value="${jdbc.password}"></property>
    </bean>
    <!--3. 创建session工厂，此类代替mybatis原来的SqlSessionFactory类的创建-->
    <bean id="sessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"></property>
        <!-- 可以省略，配置日志 -->
        <property name="configurationProperties">
            <props>
                <prop key="logImpl" >STDOUT_LOGGING</prop>
            </props>
        </property>
    </bean>

    <!--4.通过此类创建dao接口的代理实现类，代替SqlMapConfig的mappers标签-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.itheima.dao"></property>
    </bean>
    <!--5. 扫描包，通过SpringIOC创建对象-->
    <context:component-scan base-package="com.itheima.service"></context:component-scan>


    <!--6. 配置事务管理类-->
    <bean id="txManager"
                class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <!--7. 事务的通知增强-->
    <tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
            <tx:method name="find*" read-only="true"/>
            <tx:method name="*"></tx:method>
        </tx:attributes>
    </tx:advice>

    <!--8 .aop配置: 织入-->

    <aop:config>
```

```
        <aop:advisor advice-ref="txAdvice" pointcut="execution(* com.itheima.service.impl.*.*
(..))"></aop:advisor>
    </aop:config>
```

7、测试Dao层

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class TestDao {

    @Autowired
    AccountDao accountDao;

    @Test
    public void testFindAll(){
        List<Account> accountList = accountDao.findAll();

        for (Account account : accountList) {
            System.out.println(account.getUsername());
        }
    }

}
```

8、测试Service层

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class TestService {

    @Autowired
    AccountService accountService;

    @Test
    public void test(){
        accountService.transfer("zhangsan","maliu",1.0);
    }
}
```

9、创建web的控制器类

```
package com.itheima.controller;

import com.itheima.domain.Account;
import com.itheima.service.AccountService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
```

```java
import java.util.List;
/**
 * 账户的控制器类
 */
@Controller
@RequestMapping("/account")
public class AccountController {

    @Autowired
    AccountService accountService;

    /**
     * 查询全部
     * @return
     */
    @RequestMapping("/findAll")
    public ModelAndView findAll(){

        List<Account> accountList = accountService.findAll();
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("accountList", accountList);
        modelAndView.setViewName("show");
        return modelAndView;
    }

    /**
     * 删除账户
     * @param id
     * @return
     */
    @RequestMapping("/del")
    public String del(Integer id){
        accountService.deleteById(id);
        return "redirect:/account/findAll";
    }

    /**
     * 更新数据回显
     * @param id
     * @return
     */
    @RequestMapping("/updateUI")
    public ModelAndView updateUI(Integer id){
        Account account = accountService.updateAccountUI(id);
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("account", account);
        modelAndView.setViewName("account-update");
        return modelAndView;
    }
    /**
     * 更新账户
     * @param account

     * @return
```

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

10、编写spring-mvc配置文件

```
<?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/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/mvc http://www.springframework.org/schema/mvc/spring-
mvc.xsd">
    <!--1. 静态资源放行-->
    <mvc:resources mapping="/js/**" location="/js/"></mvc:resources>
    <mvc:resources mapping="/css/**" location="/css/"></mvc:resources>
    <mvc:resources mapping="/fonts/**" location="/fonts/"></mvc:resources>
    <!--2. 启动注解驱动，加载springmvc的丰富功能-->
    <mvc:annotation-driven></mvc:annotation-driven>
    <!--3. 扫描控制层的类的创建-->
    <context:component-scan base-package="com.itheima.controller"></context:component-scan>
    <!--4. 视图解析器-->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/pages/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

11、编写web.xml

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

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

  <!--指定spring主配置文件的路径，默认是/WEB-INF/applicationContext.xml-->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
  </context-param>
  <!--编码过滤器-->
  <filter>
    <filter-name>characterEncoding</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>characterEncoding</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <!--配置spring容器的监听器，负责监听tomcat的启动，创建spring容器，加载配置文件-->
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
    <!--springMVC的前端控制器-->
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
    <!--启动tomcat立即创建该servlet-->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <!--拦截所有的请求-->
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

12、编写页面

1. 引入bootStrap的资源
   见资料
   使用bootStrap的网址
       https://www.w3cschool.cn/bootstrap/
2. static.jsp-- 此文件作为公共的文件，可以在页面中使用<%@ include file="static.jsp"%>引入内容

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

```
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
    <!-- 引入CSS样式 -->
    <link rel="stylesheet" href="${pageContext.request.contextPath}/css/bootstrap.min.css">
    <!-- 引入JS文件 -->
    <script type="text/javascript" src="${pageContext.request.contextPath}/js/jquery-1.9.1.js">
</script>
    <script type="text/javascript" src="${pageContext.request.contextPath}/js/bootstrap.min.js">
</script>
3. show.jsp --- 展示数据
    <table class="table" align="center" style="width: 800px">
    <caption><a class="btn btn-primary btn-sm"
        href="${pageContext.request.contextPath}/pages/account-add.jsp">添加</a></caption>
    <thead>
    <tr>
        <th>id</th>
        <th>用户名</th>
        <th>余额</th>
        <th>操作</th>
    </tr>
    </thead>
    <tbody>
    <c:forEach items="${accountList}" var="account">
        <tr class="active">
            <td>${account.id}</td>
            <td>${account.username}</td>
            <td>${account.money}</td>
            <td>
                <a class="btn btn-primary btn-sm" href="javascript:del(${account.id})">删除</a>
                <a class="btn btn-primary btn-sm"
          href="${pageContext.request.contextPath}/account/updateUI?id=${account.id}">修改</a>
            </td>
        </tr>
    </c:forEach>
    </tbody>
</table>
</body>
<script type="text/javascript">
    function del(id){
        alert(id);
        if(confirm("确定要删除吗?")){
            location.href="${pageContext.request.contextPath}/account/del?id="+id;
        }
    }
</script>
4. account-update.jsp    -- 更新页面
<form style="width: 500px"  class="form-horizontal" role="form"
action="${pageContext.request.contextPath}/account/update">
    <!--使用隐藏域，在更新到数据库时必须使用-->
    <input type="hidden" name="id" value="${account.id}">
    <div class="form-group">
        <label for="firstname" class="col-sm-2 control-label">用户名</label>

        <div class="col-sm-10">
```

```
            <input type="text" class="form-control" name="username" id="firstname"
value="${account.username}"
                   placeholder="请输入名字">
        </div>
    </div>
    <div class="form-group">
        <label for="lastname" class="col-sm-2 control-label">余额</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" name="money" id="lastname"
value="${account.money}"
                   placeholder="请输入姓">
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <button type="submit" class="btn btn-default">更新</button>
        </div>
    </div></form>
</body>
```

5. 保存页面

```
    <form style="width: 500px"  class="form-horizontal" role="form"
action="${pageContext.request.contextPath}/account/save">
    <div class="form-group">
        <label for="firstname" class="col-sm-2 control-label">名字</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" name="username" id="firstname"
value="${account.username}"
                   placeholder="请输入名字">
        </div>
    </div>
    <div class="form-group">
        <label for="lastname" class="col-sm-2 control-label">姓</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" name="money" id="lastname"
value="${account.money}"
                   placeholder="请输入姓">
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <button type="submit" class="btn btn-default">更新</button>
        </div>
    </div></form>
```

三、拦截器

1、拦截器的作用

a. 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。
b. 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。
c. 拦截器，过滤器，监听器的区别
    过滤器：是servlet的一部分，任何web项目都可以使用
        配置 /* 后会过滤所有的资源（请求）
    拦截器：是springMVC的一部分，只能在springMVC中使用
        配置了/* 只会拦截请求，不会拦截静态资源
    监听器：Web监听器是Servlet规范中的一种特殊类，用于监听ServletContext、HttpSession和ServletRequest等域对象的创建与销毁事件，当Web应用启动时启动，当Web应用销毁时销毁。用于监听域对象的属性发生修改的事件，可以在事件发生前、发生后做一些必要的处理
d. 底层采用的是aop的思想

2、拦截器的代码

a. 声明类实现HandlerInterceptor接口
b. 自定义拦截器

```
    import org.springframework.lang.Nullable;
    import org.springframework.web.servlet.HandlerInterceptor;
    import org.springframework.web.servlet.ModelAndView;

    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;

    /**
     * 自定义拦截器
     */
    public class MyIntercepter implements HandlerInterceptor {

        /**
         * 何时访问
         *      在请求时访问时，访问业务方法前执行
         * 有什么用
         *      对所有的请求会发生拦截，对拦截的方法进行业务判断，
         *      如果需要继续业务，则返回true，放行，执行下一个拦截器，或者继续执行业务
         *      如果不需要执行业务，返回false，所有的终止
         *  执行顺序
         *      按照配置的顺序执行
         * @param request
         * @param response
         * @param handler
         * @return
         * @throws Exception
         */
        @Override
        public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler)
                throws Exception {
            System.out.println("preHandle执行了");
            return true;
        }
```

```
        /**
         * 何时访问
         *      在所有的拦截器都返回true时才会执行
         *      在controller执行完后执行，但是还没有通过视图解析器渲染页面
         * 有什么用
         *      可以对controller返回的数据进行判断处理
         * 执行顺序
         *      按照配置顺序的倒序执行
         * @param request
         * @param response
         * @param handler
         * @param modelAndView
         * @throws Exception
         */
        @Override
        public void postHandle(HttpServletRequest request, HttpServletResponse response, Object
handler,
                               @Nullable ModelAndView modelAndView) throws Exception {
            System.out.println("postHandle执行了");
        }

        /**
         * 何时调用
         *      所有的拦截器preHandle返回true时才会执行
         * 有什么用
         *      在页面渲染完成后才会执行，一般用于清理资源使用
         * 执行顺序
         *      按照配置顺序的倒序执行
         * @param request
         * @param response
         * @param handler
         * @param ex
         * @throws Exception
         */
        @Override
        public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
Object handler,
                                    @Nullable Exception ex) throws Exception {
            System.out.println("afterCompletion执行了");
        }
    }
```

c. 拦截器的配置

```
    <mvc:interceptors>
        <mvc:interceptor>
            <!-- 拦截的路径 -->
            <mvc:mapping path="/**"/>
            <!--指定定义的拦截器-->
            <bean class="com.itheima.interceptor.MyIntercepter2"></bean>
        </mvc:interceptor>
    </mvc:interceptors>
```

d. 拦截器测试

```
    preHandle执行了

    执行了自定义控制器的方法
```

```
postHandle执行了
afterCompletion执行了
```

3、多个拦截器测试

```
a. 自定义拦截器
    与上方一致
b. 配置拦截器
    <mvc:interceptors>
        <mvc:interceptor>
            <!-- 拦截的路径 -->
            <mvc:mapping path="/**"/>
            <!--指定定义的拦截器-->
            <bean class="com.itheima.interceptor.MyIntercepter"></bean>
        </mvc:interceptor>
        <mvc:interceptor>
            <!-- 拦截的路径 -->
            <mvc:mapping path="/**"/>
            <!--指定定义的拦截器-->
            <bean class="com.itheima.interceptor.MyIntercepter2"></bean>
        </mvc:interceptor>
    </mvc:interceptors>
c. 测试
    拦截器1 ： preHandle执行了
    拦截器2 ： preHandle执行了
    执行了自定义控制器的方法
    拦截器2 ： postHandle执行了
    拦截器1 ： postHandle执行了
    拦截器2 ： afterCompletion执行了
    拦截器1 ： afterCompletion执行了
```

4、在ssm中使用拦截器示例

```
a. 定义登录页面
   <form class="form-horizontal" role="form" method="post"
            action="${pageContext.request.contextPath}/login?aa=login">
        <div class="form-group">
            <label for="firstname" class="col-sm-2 control-label">账号</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="firstname" name="username"
                       placeholder="请输入账号">
            </div>
        </div>
        <div class="form-group">
            <label for="lastname" class="col-sm-2 control-label">密码</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="lastname" name="password"
                       placeholder="请输入密码">
            </div>
        </div>
        <div class="form-group">
            <div class="col-sm-offset-2 col-sm-10">
```

```
                <button type="submit" class="btn btn-default">登录</button>
            </div>
        </div>
    </form>
```

b. 定义登录的控制器

```
    @Controller
    public class LoginController {

        @RequestMapping("/login")
        public String login(HttpSession session, String username , String password){
            // 模拟在数据库中验证，如果账号密码正确登录成功
            if("zhangsan".equals(username) && "123".equals(password)){
                //如果登录成功，在session中存入username，或者存储一个user对象
                //再有请求，会从session中取出usename进行判断
                session.setAttribute("username",username);
                //登录成功，就可以访问除登录外的其他的请求了
                return "redirect:/account/findAll";
            }
            //登录失败，可以设置username为null，或者user为null
            session.setAttribute("username",null);
            //登录失败，返回登录页面
            return "redirect:/login.jsp";
        }
    }
```

c. 自定登录拦截器

```
    package com.itheima.interceptor;

    import org.springframework.lang.Nullable;
    import org.springframework.web.servlet.HandlerInterceptor;
    import org.springframework.web.servlet.ModelAndView;

    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;

    public class LoginInterceptor implements HandlerInterceptor{


        @Override
        public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler) throws Exception {
            //判断是否是登录请求，如果是，直接放行，返回true，如若不放行，此网站就被封死了
            if(request.getRequestURI().indexOf("login")>0){
                return true;
            }
           // 从session中取出登录信息
            String  username = (String) request.getSession().getAttribute("username");
            //如果登录信息不为null，说明登录成功了，放心
            if(username != null){
                return true;
            }else{
                //否则判定为没有登录，返回登录页面
                request.getRequestDispatcher("/login.jsp").forward(request,response);

                return false;
```

```
            }
        }

        @Override
        public void postHandle(HttpServletRequest request, HttpServletResponse response, Object
handler,@Nullable ModelAndView modelAndView) throws Exception {

        }

        @Override
        public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
Object handler,@Nullable Exception ex) throws Exception {

        }
    }
```