springmvc

主讲:石小俊

1.介绍springmvc

MVC框架

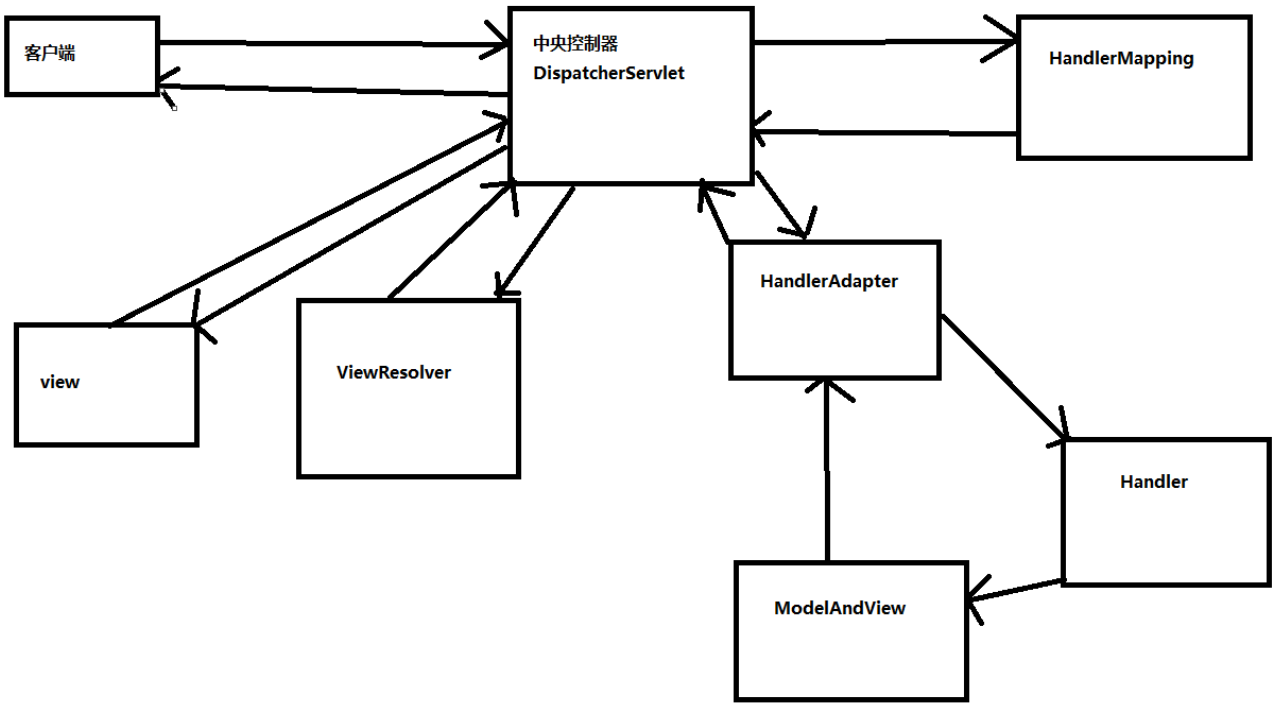
- ssh
  - spring struts1 hibernate
  - spring struts2 hibernate
  - spring springmvc hibernate
- ssm
  - spring springmvc mybatis

1-1springmvc特点

- 天生与spring整合
- 提供了大量的注解来代替原有的配置文件
- 提供了非常简单的web支持
- 提供了大量的扩展点

1-2 springmvc运行机制

```
客户端发送请求到达中央控制器DispatcherServlet
DispatcherServlet负责解析整合springmvc的运行流程
HandlerMapping负责将请求映射给不同的处理器
HandlerAdapter负责适配不同类型的处理器
Handler负责处理核心业务逻辑，并返回相应的视图与模型--相当于具体的后台代码
ModelAndView负责处理模型与视图
ViewResolver将视图结果解析为具体的视图技术
View负责具体的视图处理
最终生成响应返回给客户端
```



2.sayHello

2-1 DispatcherServlet配置

```
<!-- 配置DispatcherServlet -->
<servlet>
    <servlet-name>controller</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!-- 更改springmvc配置文件所在位置 -->
    <!--<init-param>-->
        <!--<param-name>contextConfigLocation</param-name>-->
        <!--<param-value>classpath:applicationContext.xml</param-value>-->
    <!--</init-param>-->
    <load-on-startup>0</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>controller</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

2-2 配置文件版本

```
<!-- HandlerMapping -->
<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"></bean>
```

```
<!-- HandlerAdapter -->
<bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"></bean>
<!--
   Hanlder
   此处的id是用于访问的url路径，因此需要加/
-->
<bean id="/sayHello" class="controller.HelloController"></bean>
<!-- ViewResolver -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!-- 配置具体的view -->
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"></property>
    <!-- 前缀 -->
    <property name="prefix" value="/WEB-INF/pages/"></property>
    <!-- 后缀 -->
    <property name="suffix" value=".jsp"></property>
</bean>
```

```
package controller;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

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

/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/21 9:46
 * Description:
 * version:1.0
 */
public class HelloController implements Controller {
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
throws Exception {


        ModelAndView mav = new ModelAndView();

        String name = request.getParameter("name");

        mav.addObject("msg","Hello "+name);

        mav.setViewName("hello");


        return mav;
    }
}
```

2-3 注解版

```
<context:component-scan base-package="controller"></context:component-scan>
<!--
   如果只是简单访问，可以省略
   但是在某些时候，该配置一定要有
   因此，我们在配置的时候最好加上
-->
<mvc:annotation-driven></mvc:annotation-driven>
<!-- ViewResolver -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!-- 配置具体的view -->
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"></property>
    <!-- 前缀 -->
    <property name="prefix" value="/WEB-INF/pages/"></property>
    <!-- 后缀 -->
    <property name="suffix" value=".jsp"></property>
</bean>
```

```
package controller;

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

import javax.servlet.http.HttpServletRequest;

/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/21 11:22
 * Description:
 * version:1.0
 */
@Controller
public class HelloAnnotationController {

    @RequestMapping("/hello")
    public String hello(HttpServletRequest request){
        request.setAttribute("msg","Hello World!");
        return "hello";
    }

}
```

3.响应配置

3-1 直接访问视图

```
<!--
    直接访问视图
    path:表示访问命令
    view-name:表示视图名
-->
<mvc:view-controller path="/showLogin" view-name="login"></mvc:view-controller>
```

3-2 HanIder方法返回值类型

- ModelAndView
  - 返回的是视图与模型
  - 可以单独使用视图或者模型作为结果
- String
  - 直接写字符串
    - 将当前字符串作为视图名返回
  - redirect:命令
    - 通过重定向的方式访问指定的命令
  - forward:命令
    - 通过转发的方式访问指定的命令
- void
  - 响应结果与参数有关
  - 当参数中有可以作为响应对象的参数的时候
    - 将响应结果交给响应参数来处理
  - 当参数中没有响应对象的时候
    - 将当前请求名作为视图名返回
- @ResponseBody Object
  - 只返回数据模型
  - 一般用于Ajax操作

3-3 POST请求字符编码过滤器

```
<!-- POST请求字符编码过滤器 -->
<filter>
    <filter-name>characterFilter</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>characterFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3-4 访问静态资源

默认情况下,在springmvc环境中无法直接访问静态资源

有两种方式访问静态资源

- 访问非WEB-INF下的资源

```
<mvc:default-servlet-handler></mvc:default-servlet-handler>
```

- 访问指定的资源
  - 访问的资源可以在任意位置
  - 包含WEB-INF

```
<!--
    访问指定的资源
    location属性:表示访问的资源地址
    mapping属性:表示访问命令
        *:表示的是对应的文件名，只能访问一层路径
        **:表示的是对应的文件名，可以访问多层路径
-->
<mvc:resources mapping="/img/**" location="/WEB-INF/images/"></mvc:resources>
<mvc:resources mapping="/img2/*" location="/images/"></mvc:resources>
```

4.请求配置

4-1 简单配置

使用@RequestMapping进行url地址的配置

- 当RequestMapping标注在方法上的时候
  - 表示访问该方法所需要的命令
- 当RequestMapping标注在类上的时候
  - 表示访问该类所需要的命令

4-2 ANT风格的url配置

- *
  - 匹配一层路径
- **
  - 匹配多层路径
- ？
  - 匹配一个字符
  - 同层路径下可以与其他字符串联合使用

```
@RequestMapping("/f2/*")
public void f2(){
    System.out.println("f2---*");
}

@RequestMapping("/f3/**")
public void f3(){
    System.out.println("f3---**");
}


@RequestMapping("/f4/?")
public void f4(){
    System.out.println("f4---?");
}

@RequestMapping("/f5/find?")
public void f5(){
    System.out.println("f5");
}
```

4-3 REST风格

- {xxx}
  - 只表示一层路径
  - 将该层路径具体的命令所用的字符串作为当前方法的参数所对应的值
  - 通过@PathVariable("xxx")表示给对应的哪一个参数
  - 注解中的参数可以省略，默认即为当前变量的名字

- {xxx:正则表达式}
  - 可以为当前的参数使用正则进行限制其格式

```
@RequestMapping("/f6/{id}")
public void f6(@PathVariable String id){
    System.out.println("id:"+id);
}

@RequestMapping("/f7/{id}")
public void f7(@PathVariable("id") String username){
    System.out.println("username:"+username);
}
@RequestMapping("/f8/{id:\\d+}")
public void f8(@PathVariable int id){
    System.out.println("id:"+id);
}
```

4-4 其他配置

- 同一个方法配置多个访问命令
  - 在RequestMapping中，value的值可以是字符串，也可以是数字
  - 当值为字符串的时候，value可以省略
  - 当值为数组的时候，value不可以省略

```
@RequestMapping(value={"/f9","/test9"})
public void f9(){
    System.out.println("f9");
}
```

- 可以限制请求提交方式
  - 通过method方法进行设置请求提交方式

```
@RequestMapping(value = "/f10",method = RequestMethod.POST)
public void f10(){
    System.out.println("f10");
}
```

- 限制参数
  - 限制请求中必须包含指定的参数
  - 还能限制请求中参数的值必须是什么
  - 还能限制请求中参数的值不能是什么

```
//表示请求中必须包含指定的参数
@RequestMapping(value = "/f11",params = "name")
```

```
public void f11(){
    System.out.println("f11");
}

//表示请求中必须包含指定的参数
@RequestMapping(value = "/f12",params = "name=admin")
public void f12(){
    System.out.println("f12");
}

//表示请求中必须包含指定的参数
@RequestMapping(value = "/f13",params = "name!=admin")
public void f13(){
    System.out.println("f13");
}
```

5.参数配置

- HttpServletRequest/HttpServletResponse/HttpSession
- InputStream/OutputStream
- Reader/Writer
- Model/Map/ModelMap
- @RequestParam
- @PathVariable
- @RequestBody

```
package controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.Reader;
import java.io.Writer;
import java.util.Map;
```

```java
/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/22 9:05
 * Description:
 * version:1.0
 */
@Controller
@RequestMapping("/param")
public class ParamController {

    @RequestMapping("/f1")
    public void f1(HttpServletRequest request, HttpServletResponse response){
        System.out.println("f1");
    }

    @RequestMapping("/f2")
    public void  f2(HttpSession session){
        System.out.println("f2");
    }

    @RequestMapping("/f3")
    public void f3(InputStream in, OutputStream out){
        System.out.println("f3");
    }

    @RequestMapping("/f4")
    public void f4(Reader reader, Writer writer){
        System.out.println("f4");
    }

    //Application对象不能直接作为参数
    @RequestMapping("/f5")
    public void f5(ServletContext application){
        System.out.println("f5");
    }


    @RequestMapping("/f6")
    public String f6(Model model, Map map, ModelMap modelMap){
        model.addAttribute("model","Hello Model");
        map.put("map","Hello Map");

        //ModelMap既可以作为Model使用，也可以作为Map使用
        modelMap.addAttribute("modelMap1","Hello ModelMap-Model");
        modelMap.put("modelMap2","Hello ModelMap-map");
        //默认存放在request的作用域中
        //但是与request又存在一些区别
        //重定向之后数据并不会丢失，可以通过一定的方法获取
        //详见f8
        return "model";
//        return "redirect:/showModel";
    }
```

```
    @RequestMapping("/f7")
    public String f7(){
        return "redirect:/showModel?name=admin";
    }


    @RequestMapping("/f8")
    public String f8(Model model, Map map, ModelMap modelMap){
        model.addAttribute("model","Hello Model");
        map.put("map","Hello Map");

        //ModelMap既可以作为Model使用，也可以作为Map使用
        modelMap.addAttribute("modelMap1","Hello ModelMap-Model");
        modelMap.put("modelMap2","Hello ModelMap-map");
        return "redirect:/showModel";
    }


    //通过RequestParam可以获取请求中指定的key所对应的值
    //当key的值与当前参数的变量名一致的时候，可以省略key
    //且能直接将该注解省略
    //当参数没有被其他注解所标注的时候
    //默认存在RequestParam注解
    //当key与变量名不一致的时候
    //必须通过该注解标明对应的key是谁
    @RequestMapping("/f9")
    public void f9(@RequestParam("username") String name){
        System.out.println("name:"+name);
    }


    @RequestMapping("/f10/{name}")
    public void f10(@PathVariable("name") String name){
        System.out.println("name:"+name);
    }


    @RequestMapping("/f11")
    public void f11(@RequestBody String body){
        System.out.println("-------------");
        System.out.println("body:"+body);
    }




}
```

6.访问作用域

6-1 request作用域

@ModelAttribute(key)

- 当该注解使用于参数前时
  - 该注解用于参数中时，表示默认向request作用域中读写数据
  - key表示的是存放到作用域中的数据的key
  - 当参数为一个对象，且没有使用其他注解所标注时，默认存在该注解
- 当该注解使用于方法上时
  - 表示该方法会在当前Controller中任意一个方法执行之前执行

```
package controller;

import entity.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.ArrayList;
import java.util.List;

/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/22 10:26
 * Description:
 * version:1.0
 */
@Controller
@RequestMapping("/request")
public class RequestController {

    @RequestMapping("/f1")
    public String f1(User user){
        System.out.println("f1");
        user.setUsername("admin");
        return "request";
    }
    @RequestMapping("/f2")
    public String f2(@ModelAttribute("u") User user){
        System.out.println("f2");
        user.setUsername("admin");
        return "request";
    }


    @RequestMapping("/f3")
    public void f3(User user){
        System.out.println(user.getUsername());
    }
```

```
    @ModelAttribute("list")
    public List<String> loadString(){
        System.out.println("----------------------");
        List<String> list = new ArrayList<String>();
        list.add("aaa");
        list.add("bbb");
        list.add("ccc");
        list.add("ddd");
        return list;
    }

}
```

6-2 session作用域

```
package controller;

import entity.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.bind.support.SessionStatus;

import javax.servlet.http.HttpSession;

/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/22 10:36
 * Description:
 * version:1.0
 */
@Controller
@RequestMapping("/session")
@SessionAttributes("user")
public class SessionController {

    @RequestMapping("/f1")
    public String f1(HttpSession session){
        User user = new User();
        user.setUsername("admin");
        session.setAttribute("user",user);
        return "session";
    }
```

```
    @RequestMapping("/f2")
    public void f2(HttpSession session){
        User user = (User) session.getAttribute("user");
        System.out.println(user.getUsername());
    }


    @RequestMapping("/f3")
    public void f3(@ModelAttribute("user") User user){
        System.out.println(user);
        //此时参数的对象默认是存放于request中
        //希望参数的对象默认是存放在session中
    }


    @RequestMapping("/f4")
    public void  f4(HttpSession session){
        session.invalidate();
        //当使用SessionAttributes之后
        //发现清空后数据仍然存在
        //怎么办?
    }


    @RequestMapping("/f5")
    public void f5(SessionStatus status){
        status.setComplete();
    }


}
```

6-3 application作用域

```
package controller;

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

import javax.servlet.ServletContext;
import javax.servlet.http.HttpSession;

/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/22 10:48
 * Description:
 * version:1.0
```

```
 */
@Controller
@RequestMapping("/app")
public class ApplicationController {

    @RequestMapping("/f1")
    public void f1(ServletContext application){
        System.out.println("f1");
    }

    //Application不能作为Handler方法的参数
    //那该如何获取Application对象
    @RequestMapping("/f2")
    public void f2(HttpSession session){
        ServletContext application = session.getServletContext();
        System.out.println("application:"+application);
    }

    @RequestMapping("/f3")
    public String f3(HttpSession session){
        ServletContext application = session.getServletContext();
        application.setAttribute("msg","Hello Application");
        return "application";
    }


}
```

7.异常处理

7-1 局部异常处理

```
package controller;

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

import java.io.IOException;

/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/22 11:36
 * Description:
 * version:1.0
 */
@Controller
```

```java
@RequestMapping("/exception")
public class ExceptionController {



    //@ExceptionHandler该注解表示当前方法用于处理当前Controller中的异常
    //该注解可以传递参数
    //表示当前方法处理的异常类型
//    @ExceptionHandler(SomeException.class)
//    public ModelAndView exceptionHandler(Exception e){
//        return new ModelAndView("exception","e",e);
//    }
//
//
//    @ExceptionHandler(IOException.class)
//    public ModelAndView exceptionHandler2(Exception e){
//        return new ModelAndView("exception","e",e);
//    }


    @ExceptionHandler
    public ModelAndView exceptionHandler(Exception e){
        ModelAndView mav = new ModelAndView();

        if(e instanceof SomeException){
            mav.setViewName("exception");
            mav.addObject("e",e);
        }
        if(e instanceof IOException){
            mav.setViewName("exception2");
            mav.addObject("e",e);
        }

        return mav;
    }


    @RequestMapping("/f1")
    public void f1() throws SomeException {
        throw new SomeException("SomeException");
    }


    @RequestMapping("/f2")
    public void f2() throws IOException {
        throw new IOException("IOException");
    }

}
```

7-2 全局异常处理

```
package global;

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

/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/22 13:30
 * Description:
 * version:1.0
 */
//此时必须配置<mvc:annotation-driven></mvc:annotation-driven>
@ControllerAdvice
public class GlobalExceptionHandler {


    @ExceptionHandler(SomeException.class)
    public String exceptionHandler(Exception e){
        System.out.println("正在执行全局异常处理");
        return "globalException";
    }

}
```

8.拦截器

在处理Hanlder业务方法之前或者之后以及响应到达之前做一些额外的处理

可以将于业务相关的一些操作封装成拦截器

8-1 开发步骤

- 创建一个java类
  - 该类实现HandlerInterceptor接口
  - 重写其中的三个方法

```
package interceptor;

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

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

```
/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/22 14:02
 * Description:
 * version:1.0
 */
public class LogInterceptor implements HandlerInterceptor {

    /**
     * Hanler业务方法之前进行拦截
     * @param request
     * @param response
     * @param handler
     * @return 返回值是一个布尔类型，true-放行，false-不放行
     * @throws Exception
     */
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler) throws Exception {
        System.out.println(request.getRemoteAddr()+"访问了"+handler);

        return false;
    }

    /**
     * Handler业务方法执行之后进行拦截
     * @param httpServletRequest
     * @param httpServletResponse
     * @param o
     * @param modelAndView
     * @throws Exception
     */
    public void postHandle(HttpServletRequest httpServletRequest, HttpServletResponse
httpServletResponse, Object o, ModelAndView modelAndView) throws Exception {
        System.out.println("handler业务方法处理完成");
    }

    /**
     * 整个请求处理完毕，响应到达之前处理
     * @param httpServletRequest
     * @param httpServletResponse
     * @param o
     * @param e
     * @throws Exception
     */
    public void afterCompletion(HttpServletRequest httpServletRequest, HttpServletResponse
httpServletResponse, Object o, Exception e) throws Exception {
        System.out.println("响应即将到达战场");
    }
}
```

- 配置拦截器

```
<!-- 配置拦截器 -->
<mvc:interceptors>
    <mvc:interceptor>
        <!-- 拦截谁,当遇到什么样的请求的时候需要拦截 -->
        <mvc:mapping path="/log/**"/>
        <!-- 不拦截谁,当遇到哪些请求不进行拦截,可以配置多个 -->
        <mvc:exclude-mapping path="/log/find"/>
        <mvc:exclude-mapping path="/log/add"/>
        <!-- 拦截器是谁 -->
        <bean class="interceptor.LogInterceptor"></bean>
    </mvc:interceptor>
</mvc:interceptors>
```

8-2 练习:登录检查拦截器

```
package interceptor;

import entity.User;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

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

/**
 * Author:shixiaojun@itany.com
 * Date:2018/11/22 15:26
 * Description:
 * version:1.0
 */
public class CheckLoginInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object o)
throws Exception {
        User user = (User) request.getSession().getAttribute("user");
        if(user == null){
            response.sendRedirect(request.getContextPath()+"/showLogin");
            return false;
        }
        return true;
    }

    public void postHandle(HttpServletRequest httpServletRequest, HttpServletResponse
httpServletResponse, Object o, ModelAndView modelAndView) throws Exception {

    }

    public void afterCompletion(HttpServletRequest httpServletRequest, HttpServletResponse
httpServletResponse, Object o, Exception e) throws Exception {


```

```
    }
}
```

```
<!-- 登录检查拦截器 -->
<mvc:interceptor>
    <mvc:mapping path="/**"/>
    <mvc:exclude-mapping path="/showLogin"/>
    <mvc:exclude-mapping path="/login"/>
    <bean class="interceptor.CheckLoginInterceptor"></bean>
</mvc:interceptor>
```

9.Ajax操作

9-1 返回普通字符串

```
//@ResponseBody表示当前方法返回的不再是视图
//而是数据模型
//produces:配置响应体
@ResponseBody
@RequestMapping(value = "/checkUsername",produces = "text/html;charset=utf-8")
public String checkUsername(String username){
    if("admin".equals(username)){
        return "该用户已经存在";
    }
    return "用户名可用";
}
```

9-2 返回json

```
@ResponseBody
@RequestMapping("/find")
public User find(){
    User user = new User();
    user.setId(1);
    user.setUsername("admin");
    user.setPassword("admin");
    user.setPhone("13812345678");
    user.setAddress("江苏-南京");
    user.setBirthday(new Date());
    return user;
}
```

```
private Integer id;
@JsonProperty("name")
private String username;
@JsonIgnore
private String password;
private String phone;
private String address;
@JsonFormat(pattern = "yyyy年MM月dd日 HH:mm:ss")
private Date birthday;
```

```
<jackson.version>2.9.5</jackson.version>
<!-- jackson begin -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-annotations</artifactId>
  <version>${jackson.version}</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>${jackson.version}</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>${jackson.version}</version>
</dependency>
<!-- jackson end -->
```

13.文件处理

13-1 文件上传

```
<form action="${pageContext.request.contextPath}/file/uploads" method="post"
enctype="multipart/form-data">
    用户名:<input type="text" name="username"/><br/>
    文件1:<input type="file" name="file"/><br/>
    文件2:<input type="file" name="file"/><br/>
    <input type="submit" value="提交">
</form>
```

```
<!--
    文件上传
    id必须是multipartResolver
 -->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="defaultEncoding" value="utf-8"></property>
</bean>
```

```
@RequestMapping("/upload")
public void upload(@RequestParam("file") CommonsMultipartFile file, HttpSession session) throws
IOException {
    System.out.println("文件名:"+file.getOriginalFilename());
    System.out.println("文件类型:"+file.getContentType());

    //获取服务器路径
    String path = session.getServletContext().getRealPath("/upload/"+new
SimpleDateFormat("yyyyMMdd").format(new Date()));

    File f = new File(path);
    f.mkdirs();
    file.transferTo(new File(path,file.getOriginalFilename()));

}

@RequestMapping("/uploads")
public void uploads(@RequestParam("file") CommonsMultipartFile[] files, HttpSession session)
throws IOException {
    //获取服务器路径
    String path = session.getServletContext().getRealPath("/upload/"+new
SimpleDateFormat("yyyyMMdd").format(new Date()));

    File f = new File(path);
    f.mkdirs();

    for(CommonsMultipartFile file : files){
        file.transferTo(new File(path,file.getOriginalFilename()));
    }

}
```

13-2 文件下载

```
//返回值必须是ResponseEntity
//且存在泛型，泛型的类型是当前文件在内存中的类型
@RequestMapping("/download")
public ResponseEntity<byte[]> download(HttpSession session) throws IOException {
    //获取文件
    String path = session.getServletContext().getRealPath("resource/commons-logging.jar");
    File file = new File(path);

    //设置头信息
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_OCTET_STREAM);
    headers.setContentDispositionFormData("attachment","commons-logging.jar");


    //ResponseEntity构造函数中包含多种
    //我们选择参数最全面的，共有三个参数
    //参数一:响应体,也就是你想要下载的文件
    //参数二:头信息
    //参数三:http状态
    return new ResponseEntity<byte[]>(
            FileUtils.readFileToByteArray(file),
                    headers,
                    HttpStatus.CREATED);
}
```

14.ssm整合

14-1 整合组件

- maven
- spring
- springmvc
- mybatis

14-2 pom配置

```
<?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.itany.nmms</groupId>
```

```
    <artifactId>nmms</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>nmms-dao</module>
        <module>nmms-service</module>
        <module>nmms-base</module>
        <module>nmms-web</module>
    </modules>

    <properties>
        <spring.version>4.3.5.RELEASE</spring.version>
        <mysql.version>5.1.46</mysql.version>
        <servlet.version>3.1.0</servlet.version>
        <jsp.version>2.2</jsp.version>
        <jstl.version>1.2</jstl.version>
        <jackson.version>2.9.5</jackson.version>
    </properties>

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

    <!-- AOP begin -->
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjrt</artifactId>
        <version>1.8.9</version>
    </dependency>

    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.8.9</version>
    </dependency>

    <dependency>
        <groupId>aopalliance</groupId>
        <artifactId>aopalliance</artifactId>
        <version>1.0</version>
    </dependency>

    <!-- AOP end -->

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

    <dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>1.4</version>
    </dependency>

    <!-- J2EE begin -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>${servlet.version}</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
```

```
        <artifactId>jsp-api</artifactId>
        <version>${jsp.version}</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>${jstl.version}</version>
    </dependency>

    <!-- J2EE end -->


    <!-- jackson begin -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>${jackson.version}</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>${jackson.version}</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>${jackson.version}</version>
    </dependency>
    <!-- jackson end -->


    <!-- 文件上传 -->
    <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>2.5</version>
    </dependency>


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

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

    </dependencies>

</project>
```

14-3 配置文件

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

    <!-- 扫包 -->
    <context:component-scan base-package="com.itany.nmms.service.impl"></context:component-scan>


    <!-- 访问properties文件 -->
    <context:property-placeholder location="classpath:dataSource.properties"></context:property-
placeholder>

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

    <!-- 配置SqlSessionFactory -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- 注入数据源 -->
        <property name="dataSource" ref="dataSource"></property>
        <!-- 配置别名 -->
        <property name="typeAliasesPackage" value="com.itany.nmms.entity"></property>
        <!-- 注册mapper文件 -->
        <property name="mapperLocations">
```

```
            <list>
                <value>classpath:com.itany.nmms.mapper/*.xml</value>
            </list>
        </property>
    </bean>

    <!-- 配置dao接口 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.itany.nmms.dao"></property>
    </bean>

    <!-- 事务配置 -->
    <bean id="transcationManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <!-- 配置注解事务 -->
    <tx:annotation-driven transaction-manager="transcationManager"></tx:annotation-driven>


</beans>
```

- web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
          http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">


    <!-- 配置spring容器位置 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext*.xml</param-value>
    </context-param>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>


    <!-- 配置DispatcherServlet -->
    <servlet>
        <servlet-name>controller</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>0</load-on-startup>
    </servlet>
    <servlet-mapping>
```

```
        <servlet-name>controller</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- Post请求字符编码过滤器 -->
    <filter>
        <filter-name>character</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>character</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

- controller-servlet.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/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">


    <!-- 扫包 -->
    <context:component-scan base-package="com.itany.nmms.controller"></context:component-scan>


    <!-- HandlerMapping+HandlerAdapter -->
    <mvc:annotation-driven></mvc:annotation-driven>

    <!-- ViewResolver -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView">
</property>
        <property name="prefix" value="/WEB-INF/pages/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

    <!-- 访问静态资源 -->
```

```
    <mvc:default-servlet-handler></mvc:default-servlet-handler>

    <!-- 配置直接访问视图 -->
    <mvc:view-controller path="/showLogin" view-name="backend/login"></mvc:view-controller>

</beans>
```