**Spring与springMVC总结**

**1. 为什么使用Spring ?**

    1). 方便解耦，简化开发

         通过Spring提供的IoC容器，可以将对象之间的依赖关系交由Spring进行控制，避免硬编码所造成的过度程序耦合。

    2). AOP编程的支持

         通过Spring提供的AOP功能，方便进行面向切面的编程，如性能监测、事务管理、日志记录等。

    3). 声明式事务的支持

    4). 方便集成各种优秀框架

    5). 降低Java EE API的使用难度

         如对JDBC，JavaMail，远程调用等提供了简便封装

**2. 什么是IoC，为什使用IoC ?**

    IoC全称Iversion of Controller，控制反转。

**这概念是说你不用创建对象，而只需要描述它如何被创建。**你不在代码里直接组装你的组件和服务，但是要在配置文件里描述哪些组件需要哪些服务，之后一个容器（IOC容器）负责把他们组装起来。

    它能指导我们如何设计出松耦合、更优良的程序。

**3. 什么是AOP，为什么使用AOP ?**

    AOP全称：Aspect-Oriented Programming，面向切面编程。

     AOP，面向切面编程，就是把可重用的功能提取出来，然后将这些通用功能在合适的时候织入到应用程序中，比如**事务管理**、**权限控制**、**日志记录**、性能统计等。

**AOP并没有帮助我们解决任何新的问题，它只是提供了一种更好的办法，能够用更少的工作量来解决现有的一些问题，使得系统更加健壮，可维护性更好。**

**4. 什么是Spring的事务管理？**

    事务就是对一系列的数据库操作（比如插入多条数据）进行统一的提交或回滚操作，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之前的所有操作。这样可以防止出现脏数据，防止数据库数据出现问题。

    开发中为了避免这种情况一般都会进行事务管理。

**Spring的声明式事务通常是指在配置文件中对事务进行配置声明**，其中包括了很多声明属性，它是通过Spring Proxy帮你做代理，自己不用额外的写代码，只要在Spring配置文件中声明即可；通常用在数据库的操作里面；

**编程式事务就是指通过硬编码的方式做事务处理**，这种处理方式需要写代码，事务中的逻辑可以自己定制；可以是数据库的东东，也可以是其他的操作。

**Spring中也有自己的事务管理机制，一般是使用TransactionMananger进行管理，**可以通过Spring的注入来完成此功能。

**5. Spring框架支持以下五种bean的作用域：**

**singleton : 默认值，bean在每个Spring ioc 容器中只有一个实例。**

    prototype：一个bean的定义可以有多个实例。

    request：每次http请求都会创建一个bean，该作用域仅在基于web的Spring ApplicationContext情形下有效。

    session：在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

    global-session：在一个全局的HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

**6. 什么是Spring的MVC框架？**

    Spring 配备构建Web 应用的全功能MVC框架。Spring可以很便捷地和其他MVC框架集成，如Struts，Spring 的MVC框架用控制反转把业务对象和控制逻辑清晰地隔离。它也允许以声明的方式把请求参数和业务对象绑定。

    spring mvc是一个基于mvc的web框架。spring mvc是spring框架的一个模块，springmvc和spring无需通过中间整合层进行整合。

**7. 如何启用注解：**

    <context:annotation-config/>

**如果使用<context:component-scan base-package="com.tgb.web.controller.annotation"> </context:component-scan>  则上面内容可以省略**

**8. Spring MVC的请求流程：**

    第一步：发起请求到**前端控制器**(DispatcherServlet)

    第二步：前端控制器请求**HandlerMapping**查找Handler可以根据xml配置、注解进行查找

    第三步：处理器映射器HandlerMapping向前端控制器返回**Handler**

    第四步：前端控制器调用**处理器适配器**去执行Handler

    第五步：处理器适配器去执行Handler

    第六步：Handler执行完成给适配器返回**ModelAndView**

    第七步：处理器适配器向前端控制器返回ModelAndView。ModelAndView是springmvc框架的一个底层对象，包括 Model和view

    第八步：前端控制器请求**视图解析器**去进行视图解析，根据逻辑视图名解析成真正的视图(jsp)

    第九步：视图解析器向前端控制器**返回View**

    第十步：前端控制器进行**视图渲染**。视图渲染将模型数据(在ModelAndView对象中)填充到request域

    第十一步：前端控制器向用户响应结果

**9. web.xml的配置**



**10. 注解的处理器映射器和适配器**

      spring3.1之后使用org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping注解映射器。

      在spring3.1之后使用org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter注解适配器。

**使用 mvc:annotation-driven代替上边注解映射器和注解适配器配置**

**11. spring 与 mybatis整合过程**

      第一步：整合dao层

         mybatis和spring整合，通过spring管理mapper接口。

         使用mapper的扫描器自动扫描mapper接口在spring中进行注册。

      第二步：整合service层

         通过spring管理 service接口。

         使用配置方式将service接口配置在spring配置文件中。

         实现事务控制。

      第三步：整合springmvc

         由于springmvc是spring的模块，不需要整合。

**主要配置有：**

      1). mybatis配置文件sqlMapConfig.xml配置别名自动扫描**(实体类)**

      2). mapper扫描器**(接口，数据库访问接口)**

      3). 数据库连接池配置

      4). 声明式事务配置

      5). 启用注解扫描：<context:component-scan base-package="cn.itcast.ssm.controller"></context:component-scan>

      6). 配置注解映射器和适配器： <mvc:annotation-driven></mvc:annotation-driven>

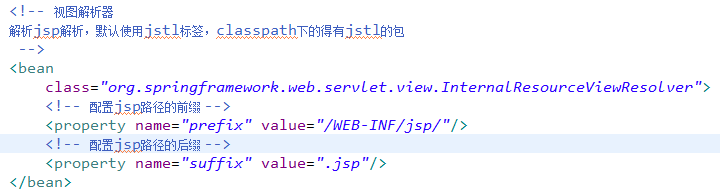
      7). 视图解析器：<bean  class="org.springframework.web.servlet.view.InternalResourceViewResolver">

      8). 配置控制类： DispatcherServlet前端控制器

      9). 配置spring配置文件加载类：ClassLoadListener

**12.** 前端控制器从上边的文件中加载处理映射器、适配器、视图解析器等组件，如果不在springmvc.xml中配置，使用默认加载的 DispatcherSerlvet.properties。

**13. 视图解析器配置前缀和后缀：**



**14. sqlMapConfig.xml，mybatis自己的配置文件。**



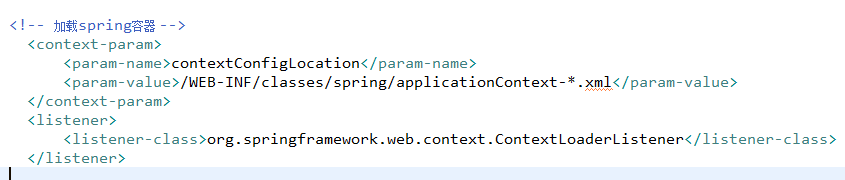
**15. 配置数据源：**



**16. 事务控制(applicationContext-transaction.xml)**，在applicationContext-transaction.xml中使用spring声明式事务控制方法。



**17. 加载spring配置**



**18. 静态资源访问不被拦截：**

      <resources mapping="/resources/\*\*" location="/resources/" />

      <resources mapping="/images/\*\*" location="/images/" />

      <resources mapping="/js/\*\*" location="/js/" />

**19. @RequestMapping的作用**

      1). url映射

      2). 窄化请求映射

      3). 限制http请求方法

**20. controller方法的返回值**

**20.1 返回ModelAndView**

              需要方法结束时，定义ModelAndView，将model和view分别进行设置。

**20.2 返回string**

               如果controller方法返回string，

               1). 表示返回逻辑视图名。真正视图(jsp路径)=前缀+逻辑视图名+后缀

               2). redirect重定向：返回字符串格式为："redirect:queryItem.action"

               3). forward页面转发：返回字符串格式为：“forward:queryItem.action”

**20.3 返回void**

               在controller方法形参上可以定义request和response，使用request或response指定响应结果：

               1). 使用request转向页面，如下：request.getRequestDispatcher("页面路径").forward(request, response);

               2). 也可以通过response页面重定向：response.sendRedirect("url")

               3). 也可以通过response指定响应结果，例如响应json数据如下：

                    response.setCharacterEncoding("utf-8");

                    response.setContentType("application/json;charset=utf-8");

                    response.getWriter().write("json串");

**21. 参数绑定：**

**21.1 默认支持的类型**

               直接在controller方法形参上定义下边类型的对象，就可以使用这些对象。在参数绑定过程中，如果遇到下边类型直接进行绑定。

               1). HttpServletRequest：通过request对象获取请求信息

               2). HttpServletResponse：通过response处理响应信息

               3). HttpSession：通过session对象得到session中存放的对象

               4). Model/ModelMap：model是一个接口，modelMap是一个接口实现 。作用：将model数据填充到request域。

**21.2 简单类型**

              通过@RequestParam对简单类型的参数进行绑定。

**如果不使用@RequestParam，要求request传入参数名称和controller方法的形参名称一致，方可绑定成功。**

**如果使用@RequestParam，不用限制request传入参数名称和controller方法的形参名称一致。**

              通过required属性指定参数是否必须要传入，如果设置为true，没有传入参数，会报错。

**21.3 pojo绑定**

**页面中input的name和controller的pojo形参中的属性名称一致，将页面中数据绑定到pojo。(usename,age；不需要user.username,user.age)**

**21.4 自定义参数绑定实现日期类型绑定**

              对于controller形参中pojo对象，如果属性中有日期类型，需要自定义参数绑定。将请求日期数据串转成 日期类型，要转换的日期类型和pojo中日期属性的类型保持一致。

**22. Spring MVC 和 Struts2 对比**

**1). Struts2是类级别的拦截， 一个类对应一个request上下文，SpringMVC是方法级别的拦截，一个方法对应一个request上下文，而方法同时又跟一个url对应，所以说从架构本身上SpringMVC 就容易实现restful url**

      2). 由上边原因，SpringMVC的方法之间基本上独立的，独享request response数据，请求数据通过参数获取，处理结果通过ModelMap交回给框架，方法之间不共享变量，而Struts2搞的就比较乱，虽然方法

           之间也是独立的，但其所有Action变量是共享的，这不会影响程序运行，却给我们编码 读程序时带来麻烦，每次来了请求就创建一个Action，一个Action对象对应一个request上下文。

**3). 由于Struts2需要针对每个request进行封装，把request，session等servlet生命周期的变量封装成一个一个Map，供给每个Action使用，并保证线程安全，所以在原则上，是比较耗费内存的。**

      4). SpringMVC集成了Ajax，使用非常方便，只需一个注解@ResponseBody就可以实现，然后直接返回响应文本即可，而Struts2拦截器集成了Ajax，在Action中处理时一般必须安装插件或者自己

           写代码集成进去，使用起来也相对不方便。

      5). springmvc面向方法开发的（更接近service接口的开发方式），struts2面向类开发。

**6). springmvc可以单例开发，struts2只能是多例开发。**

**23. 乱码处理**

      1). post乱码

**在web.xml添加post乱码filter：CharacterEncodingFilter**

      2). 对于get请求中文参数出现乱码解决方法有两个：

           a. 修改tomcat配置文件添加编码与工程编码一致，如下：

               <Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

           b. 对参数进行重新编码：

               String userName = new  String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")

               ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码

**24. 集合类型绑定**

      1). 数组绑定：

           controller方法参数使用：(Integer[] itemId)

           页面统一使用:itemId 作为name

      2). list绑定：

           pojo属性名为：itemsList

           页面：itemsList[index].属性名

      3). map 绑定：

           pojo属性名为：Map<String, Object> itemInfo = new HashMap<String, Object>();

           页面： <td>姓名：<inputtype="text"name="itemInfo[‘name‘]"/>

**25. spring 校验**

      1). 项目中，通常使用较多是前端的校验，比如页面中js校验。对于安全要求较高点建议在服务端进行校验。

      2). springmvc使用hibernate的校验框架validation(和hibernate没有任何关系)。

           校验思路：页面提交请求的参数，请求到controller方法中，使用validation进行校验。如果校验出错，将错误信息展示到页面。

**26. 数据回显**

**1). @ModelAttribute还可以将方法的返回值传到页面：在方法上加注解@ModelAttribute**

**2). 使用最简单方法使用model，可以不用@ModelAttribute：model.addAttribute("id", id);**

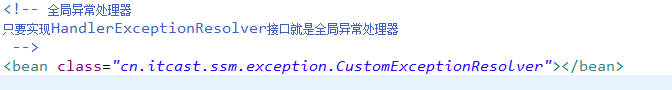
      3). springmvc默认对pojo数据进行回显。pojo数据传入controller方法后，springmvc自动将pojo数据放到request域，key等于pojo类型（首字母小写）

**4). public String testParam(PrintWriter out, @RequestParam("username") String username) { //out直接输出**

**27. 异常处理**

      springmvc提供全局异常处理器（一个系统只有一个异常处理器）进行统一异常处理。

**系统遇到异常，在程序中手动抛出，dao抛给service、service给controller、controller抛给前端控制器，前端控制器调用全局异常处理器。**



**28. 上传图片**

      1). 在页面form中提交enctype="multipart/form-data"的数据时，需要springmvc对multipart类型的数据进行解析。

      2). 在springmvc.xml中配置multipart类型解析器。



**3). 方法中使用：MultipartFile attach (单个文件上传) 或者  MultipartFile[] attachs (多个文件上传)**

**29. Json处理**

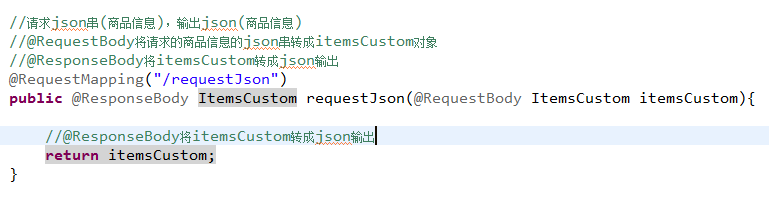
       1). 加载json转换的jar包：springmvc中使用jackson的包进行json转换（@requestBody和@responseBody使用下边的包进行json转）

       2). 配置json转换器。在注解适配器RequestMappingHandlerAdapter中加入messageConverters。**如果使用<mvc:annotation-driven /> 则会自动加入。**

       3). ajax



       4). Controller**(ResponseBody、RequestBody)**



       5). 注意ajax中contentType如果不设置为json类型，则传的参数为key/value类型。**上面设置后，传的是json类型。**

             详见：<http://www.cnblogs.com/Jtianlin/p/5224885.html>

**30. 拦截器：**

**1). 定义拦截器，实现HandlerInterceptor接口。接口中提供三个方法。**

            a. preHandle ：进入 Handler方法之前执行，用于身份认证、身份授权，比如身份认证，如果认证通过表示当前用户没有登陆，需要此方法拦截不再向下执行

            b. postHandle：**进入Handler方法之后，返回modelAndView之前执行**，应用场景从modelAndView出发：将公用的模型数据(比如菜单导航)在这里传到视图，也可以在这里统一指定视图

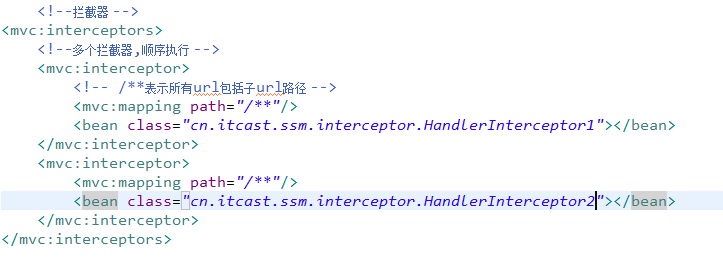
            c. afterCompletion：执行Handler完成执行此方法，应用场景：统一异常处理，统一日志处理

      2). 拦截器配置：

           a. 针对HandlerMapping配置(不推荐)：springmvc拦截器针对HandlerMapping进行拦截设置，如果在某个HandlerMapping中配置拦截，经过该 HandlerMapping映射成功的handler

               最终使用该 拦截器。  (一般不推荐使用)

**b. 类似全局的拦截器：springmvc配置类似全局的拦截器，springmvc框架将配置的类似全局的拦截器注入到每个HandlerMapping中**



31. **如何启用注解：**

      <context:annotation-config/>

**如果使用<context:component-scan base-package="com.tgb.web.controller.annotation"> </context:component-scan>  则上面内容可以省略**

**使用 mvc:annotation-driven代替注解映射器和注解适配器配置**