参考：<http://www.cnblogs.com/caolei1108/p/6196743.html>

[**springmvc配置文件web.xml详解各方总结。**](http://www.cnblogs.com/caolei1108/p/6196743.html)

Spring分为多个文件进行分别的配置，其中在servlet-name中如果没有指定init-param属性，那么系统自动寻找的spring配置文件为[servlet-name]-servlet.xml。  
当需要载入多个spring相关的配置文件时，首先加载ContextLoaderListener类，再指定context-param中指定多个spring配置文件，使用逗号分别隔开各个文件。为了使用方便可以将配置文件进行MVC式的分解，配置控制器Bean的配置文件放置在一个xml文件中，server的Bean放在service.xml文件中。

<servlet-mapping>指定的该servlet接管的url的行为，此处为了简便起见使用\*.\*,则表示在URL只要是在本机使用的任何request都是由该dispatchServlet来处理。

1. <?xml version="1.0"encoding="UTF-8"?>
2. <web-app id="WebApp\_ID" version="2.4"xmlns="http://java.sun.com/xml/ns/j2ee"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"xsi:schemaLocation="http://java.sun.com/xml/ns/j2eehttp://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">
3. <listener>
4. <listener-**class>org.springframework.web.context.ContextLoaderListener</listener-class>**
5. </listener>
6. <context-param>
7. <param-name>contextConfigLocation</param-name>
8. <param-value>/WEB-INF/applicationContext.xml,/WEB-INF/controllers.xml,/WEB-INF/service.xml</param-value>
9. </context-param>
10. <servlet>
11. <servlet-name>dispatch</servlet>
12. <servlet-**class>org.springframework.web.servlet.DispatcherServlet</servlet-class>**
13. <init-param>
14. <param-name>contextConfigLocation</param-name>
15. <param-value>/WEB-INF/applicationContext.xml</param-value>
16. </init-param>
17. </servlet>
18. <servlet-mapping>
19. <servlet-name>dispatch</servlet-name>
20. <servlet-pattern>\*.\*</servlet-pattern>
21. </servlet-mapping>
22. </web-app>

第二：

现在主流的Web MVC框架除了Struts这个主力 外，其次就是[spring](http://lib.csdn.net/base/javaee) MVC了，因此这也是作为一名程序员需要掌握的主流框架，框架选择多了，应对多变的需求和业务时，可实行的方案自然就多了。不过要想灵活运用Spring MVC来应对大多数的Web开发，就必须要掌握它的配置及原理。

　　一、Spring MVC环境搭建：（Spring 2.5.6 + [hibernate](http://lib.csdn.net/base/javaee) 3.2.0）

　　1. jar包引入

　　Spring 2.5.6：spring.jar、spring-webmvc.jar、commons-logging.jar、cglib-nodep-2.1\_3.jar

　　Hibernate 3.6.8：hibernate3.jar、hibernate-jpa-2.0-api-1.0.1.Final.jar、antlr-2.7.6.jar、commons-collections-3.1、dom4j-1.6.1.jar、javassist-3.12.0.GA.jar、jta-1.1.jar、slf4j-api-1.6.1.jar、slf4j-nop-1.6.4.jar、相应[数据库](http://lib.csdn.net/base/mysql)的驱动jar包

SpringMVC是一个基于DispatcherServlet的MVC框架，每一个请求最先访问的都是DispatcherServlet，DispatcherServlet负责转发每一个Request请求给相应的Handler，Handler处理以后再返回相应的视图(View)和模型(Model)，返回的视图和模型都可以不指定，即可以只返回Model或只返回View或都不返回。

DispatcherServlet是继承自HttpServlet的，既然SpringMVC是基于DispatcherServlet的，那么我们先来配置一下DispatcherServlet，好让它能够管理我们希望它管理的内容。HttpServlet是在web.xml文件中声明的。

<!-- Spring MVC配置 -->

<!-- ====================================== -->

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 可以自定义servlet.xml配置文件的位置和名称，默认为WEB-INF目录下，名称为[<servlet-name>]-servlet.xml，如spring-servlet.xml

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring-servlet.xml</param-value>&nbsp; 默认

</init-param>

-->

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

<!-- Spring配置 -->

<!-- ====================================== -->

<listener>

<listenerclass>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<!-- 指定Spring Bean的配置文件所在目录。默认配置在WEB-INF目录下 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:config/applicationContext.xml</param-value>

</context-param>

 spring-servlet.xml配置

　　spring-servlet这个名字是因为上面web.xml中<servlet-name>标签配的值为spring（<servlet-name>spring</servlet-name>），再加上“-servlet”后缀而形成的spring-servlet.xml文件名，如果改为springMVC，对应的文件名则为springMVC-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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd"http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<!-- 启用spring mvc 注解 -->

<context:annotation-config />

<!-- 设置使用注解的类所在的jar包 -->

<context:component-scan base-package="controller"></context:component-scan>

<!-- 完成请求和注解POJO的映射 -->

<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />

<!-- 对转向页面的路径解析。prefix：前缀， suffix：后缀 -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" p:prefix="/jsp/" p:suffix=".jsp" />

</beans>

DispatcherServlet会利用一些特殊的bean来处理Request请求和生成相应的视图返回。

关于视图的返回，Controller只负责传回来一个值，然后到底返回的是什么视图，是由视图解析器控制的，在jsp中常用的视图解析器是InternalResourceViewResovler，它会要求一个前缀和一个后缀

在上述视图解析器中，如果Controller返回的是blog/index，那么通过视图解析器解析之后的视图就是/jsp/blog/index.jsp。

主要是说说Controller.

一个类使用了@Controller进行标记的都是Controller

package controller;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;

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

import org.springframework.web.bind.annotation.RequestParam;

import entity.User;

@Controller //类似Struts的Action

public class TestController {

@RequestMapping("test/login.do") // 请求url地址映射，类似Struts的action-mapping

public String testLogin(@RequestParam(value="username")String username, String password, HttpServletRequest request) {

// @RequestParam是指请求url地址映射中必须含有的参数(除非属性required=false)

// @RequestParam可简写为：@RequestParam("username")

if (!"admin".equals(username) || !"admin".equals(password)) {

return "loginError"; // 跳转页面路径（默认为转发），该路径不需要包含spring-servlet配置文件中配置的前缀和后缀

}

return "loginSuccess";

}

@RequestMapping("/test/login2.do")

public ModelAndView testLogin2(String username, String password, int age){

// request和response不必非要出现在方法中，如果用不上的话可以去掉

// 参数的名称是与页面控件的name相匹配，参数类型会自动被转换

if (!"admin".equals(username) || !"admin".equals(password) || age < 5) {

return new ModelAndView("loginError"); // 手动实例化ModelAndView完成跳转页面（转发），效果等同于上面的方法返回字符串

}

return new ModelAndView(new RedirectView("../index.jsp")); // 采用重定向方式跳转页面

// 重定向还有一种简单写法

// return new ModelAndView("redirect:../index.jsp");

}

@RequestMapping("/test/login3.do")

public ModelAndView testLogin3(User user) {

// 同样支持参数为表单对象，类似于Struts的ActionForm，User不需要任何配置，直接写即可

String username = user.getUsername();

String password = user.getPassword();

int age = user.getAge();

if (!"admin".equals(username) || !"admin".equals(password) || age < 5) {

return new ModelAndView("loginError");

}

return new ModelAndView("loginSuccess");

}

@Resource(name = "loginService") // 获取applicationContext.xml中bean的id为loginService的，并注入

private LoginService loginService; //等价于spring传统注入方式写get和set方法，这样的好处是简洁工整，省去了不必要得代码

@RequestMapping("/test/login4.do")

public String testLogin4(User user) {

if (loginService.login(user) == false) {

return "loginError";

}

return "loginSuccess";

}

}

以上4个方法示例，是一个Controller里含有不同的请求url，也可以采用一个url访问，通过url参数来区分访问不同的方法，代码如下：

package controller;

import org.springframework.stereotype.Controller;

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

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

@RequestMapping("/test2/login.do") // 指定唯一一个\*.do请求关联到该Controller

public class TestController2 {

@RequestMapping

public String testLogin(String username, String password, int age) {

// 如果不加任何参数，则在请求/test2/login.do时，便默认执行该方法

if (!"admin".equals(username) || !"admin".equals(password) || age < 5) {

return "loginError";

}

return "loginSuccess";

}

@RequestMapping(params = "method=1", method=RequestMethod.POST)

public String testLogin2(String username, String password) {

// 依据params的参数method的值来区分不同的调用方法

// 可以指定页面请求方式的类型，默认为get请求

if (!"admin".equals(username) || !"admin".equals(password)) {

return "loginError";

}

return "loginSuccess";

}

@RequestMapping(params = "method=2")

public String testLogin3(String username, String password, int age) {

if (!"admin".equals(username) || !"admin".equals(password) || age < 5) {

return "loginError";

}

return "loginSuccess";

}

}

　其实RequestMapping在Class上，可看做是父Request请求url，而RequestMapping在方法上的可看做是子Request请求url，父子请求url最终会拼起来与页面请求url进行匹配，因此RequestMapping也可以这么写：

package controller;

import org.springframework.stereotype.Controller;

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

@Controller

@RequestMapping("/test3/\*") // 父request请求url

public class TestController3 {

@RequestMapping("login.do") // 子request请求url，拼接后等价于/test3/login.do

public String testLogin(String username, String password, int age) {

if (!"admin".equals(username) || !"admin".equals(password) || age < 5) {

return "loginError";

}

return "loginSuccess";

}

}

在SpringMVC中常用的注解还有@PathVariable，@RequestParam，@PathVariable标记在方法的参数上，利用它标记的参数可以利用请求路径传值，看下面一个例子

@RequestMapping(value="/comment/{blogId}", method=RequestMethod.POST)

public void comment(Comment comment,@PathVariable int blogId, HttpSession session, HttpServletResponse response) throws IOException {

}

在该例子中，blogId是被@PathVariable标记为请求路径变量的，如果请求的是/blog/comment/1.do的时候就表示blogId的值为1. 同样@RequestParam也是用来给参数传值的，但是它是从头request的参数里面取值，相当于request.getParameter("参数名")方法。

在Controller的方法中，如果需要WEB元素HttpServletRequest，HttpServletResponse和HttpSession，只需要在给方法一个对应的参数，那么在访问的时候SpringMVC就会自动给其传值，但是需要注意的是在传入Session的时候如果是第一次访问系统的时候就调用session会报错，因为这个时候session还没有生成。

第三：

这篇配置只是所有配置文件中集合起来的，仅仅是为了对配置文件中的bean进行说明，如果对号复制到功能中肯定是错误的。上传的文件中会有更加详细的说明！

1. Web.xml

<!--配置页面控制器-->

<servlet>

     <servlet-name>spring</servlet-name>

     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

     <init-param>

         <param-name>contextConfigLocation</param-name>

         <param-value>/WEB-INF/spring-servlet.xml</param-value>

     </init-param>

     <load-on-startup>1</load-on-startup>

 </servlet>

 <servlet-mapping>

     <servlet-name>spring</servlet-name>

     <url-pattern>/</url-pattern>

 </servlet-mapping>

 <listener>

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

 </listener>

 <!-- 指定Spring Bean的配置文件所在目录。默认配置在WEB-INF目录下 -->

 <context-param>

     <param-name>contextConfigLocation</param-name>

     <param-value>classpath:config/applicationContext.xml</param-value>

 </context-param>

 <!-- 设置字符编码方式-->

 <filter>

     <filter-name>setcharacter</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>setcharacter</filter-name>

    <url-pattern>/\*</url-pattern>

 </filter-mapping>

备注：

关于ContextLoaderListener

部署applicationContext的xml文件，如果在web.xml中不写任何参数配置信息，默认的路径是"/WEB-INF/applicationContext.xml，在WEB-INF目录下创建的xml文件的名称必须是applicationContext.xml。如果是要自定义文件名可以在web.xml里加入contextConfigLocation这个context参数：在<param-value> </param-value>里指定相应的xml文件名，如果有多个xml文件，可以写在一起并一“,”号分隔。

采用通配符，比如这那个目录下有applicationContext-ibatis-base.xml,applicationContext-action.xml，applicationContext-ibatis-dao.xml等文件，都会一同被载入。

由此可见applicationContext.xml的文件位置就可以有两种默认实现：

第一种：直接将之放到/WEB-INF下，之在web.xml中声明一个listener

第二种：将之放到classpath下，但是此时要在web.xml中加入<context-param>，用它来指明你的

1. spring - 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:p="http://www.springframework.org/schema/p"

        xmlns:context="http://www.springframework.org/schema/context"

   xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <context:annotation-config />

      <!-- 把标记了@Controller注解的类转换为bean -->

      <context:component-scan base-package="com.mvc.controller" />

  <!-- 启动Spring MVC的注解功能，完成请求和注解POJO的映射 -->

  <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />

  <!-- 对模型视图名称的解析，即在模型视图名称添加前后缀 -->

  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"

          p:prefix="/WEB-INF/view/"

          p:suffix=".jsp" />

  <bean id="multipartResolver"

          class="org.springframework.web.multipart.commons.CommonsMultipartResolver"

          p:defaultEncoding="utf-8" />

 </beans>

1. ApplicationContext.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

 xmlns:aop="http://www.springframework.org/schema/aop" xmlns:context="http://www.springframework.org/schema/context"

 xmlns:p="http://www.springframework.org/schema/p" xmlns:tx="http://www.springframework.org/schema/tx"

 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-3.0.xsd

   http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd

   http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

   http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

 <context:annotation-config />

 <context:component-scan base-package="com.mvc" />  <!-- 自动扫描所有注解该路径 -->

 <context:property-placeholder location="classpath:/hibernate.properties" />

 <bean id="sessionFactory"

  class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

  <property name="dataSource" ref="dataSource" />

  <property name="hibernateProperties">

   <props>

    <prop key="hibernate.dialect">${dataSource.dialect}</prop>

    <prop key="hibernate.hbm2ddl.auto">${dataSource.hbm2ddl.auto}</prop>

    <prop key="hibernate.hbm2ddl.auto">update</prop>

   </props>

  </property>

  <property name="packagesToScan">

   <list>

    <value>com.mvc.entity</value><!-- 扫描实体类，也就是平时所说的model -->

   </list>

    </property>

 </bean>

 <bean id="transactionManager"

  class="org.springframework.orm.hibernate3.HibernateTransactionManager">

  <property name="sessionFactory" ref="sessionFactory" />

  <property name="dataSource" ref="dataSource" />

 </bean>

 <bean id="dataSource"

  class="org.springframework.jdbc.datasource.DriverManagerDataSource">

  <property name="driverClassName" value="${dataSource.driverClassName}" />

  <property name="url" value="${dataSource.url}" />

  <property name="username" value="${dataSource.username}" />

  <property name="password" value="${dataSource.password}" />

 </bean>

 <!-- Dao的实现 -->

 <bean id="entityDao" class="com.mvc.dao.EntityDaoImpl">

  <property name="sessionFactory" ref="sessionFactory" />

 </bean>

 <tx:annotation-driven transaction-manager="transactionManager" />

 <tx:annotation-driven mode="aspectj"/>

    <aop:aspectj-autoproxy/>

</beans>

 <!-- 通过AOP配置提供事务增强，让service包下所有Bean的所有方法拥有事务 -->

 <aop:config proxy-target-class="true">

     <aop:pointcut id="serviceMethod"

         expression=" execution(\* com.service..\*(..))" />

     <aop:advisor pointcut-ref="serviceMethod" advice-ref="txAdvice" />

 </aop:config>

 <tx:advice id="txAdvice" transaction-manager="transactionManager">

     <tx:attributes>

         <tx:method name="\*" />

     </tx:attributes>

 </tx:advice>

<!-- 配置Jdbc模板 -->

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">

  <property name="dataSource" ref="dataSource"></property>

</bean>

 <!-- 配置数据源 -->

 <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"

     destroy-method="close">

     <property name="driverClass">

         <value>${jdbc.driverClassName}</value>

     </property>

     <property name="jdbcUrl">

         <value>${jdbc.url}</value>

     </property>

     <property name="user">

         <value>${jdbc.username}</value>

     </property>

     <property name="password">

         <value>${jdbc.password}</value>

     </property>

     <!--连接池中保留的最小连接数。 -->

     <property name="minPoolSize">

         <value>5</value>

     </property>

     <!--连接池中保留的最大连接数。Default: 15 -->

     <property name="maxPoolSize">

         <value>30</value>

     </property>

     <!--初始化时获取的连接数，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->

     <property name="initialPoolSize">

         <value>10</value>

     </property>

     <!--最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。Default: 0 -->

     <property name="maxIdleTime">

         <value>60</value>

     </property>

     <!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。Default: 3 -->

     <property name="acquireIncrement">

         <value>5</value>

     </property>

     <!--JDBC的标准参数，用以控制数据源内加载的PreparedStatements数量。但由于预缓存的statements 属于单个connection而不是整个连接池。所以设置这个参数需要考虑到多方面的因素。

         如果maxStatements与maxStatementsPerConnection均为0，则缓存被关闭。Default: 0 -->

     <property name="maxStatements">

         <value>0</value>

     </property>

     <!--每60秒检查所有连接池中的空闲连接。Default: 0 -->

     <property name="idleConnectionTestPeriod">

         <value>60</value>

     </property>

     <!--定义在从数据库获取新连接失败后重复尝试的次数。Default: 30 -->

     <property name="acquireRetryAttempts">

         <value>30</value>

     </property>

     <!--获取连接失败将会引起所有等待连接池来获取连接的线程抛出异常。但是数据源仍有效 保留，并在下次调用getConnection()的时候继续尝试获取连接。如果设为true，那么在尝试

         获取连接失败后该数据源将申明已断开并永久关闭。Default: false -->

     <property name="breakAfterAcquireFailure">

         <value>true</value>

     </property>

     <!--因性能消耗大请只在需要的时候使用它。如果设为true那么在每个connection提交的 时候都将校验其有效性。建议使用idleConnectionTestPeriod或automaticTestTable

         等方法来提升连接[测试](http://lib.csdn.net/base/softwaretest)的性能。Default: false -->

     <property name="testConnectionOnCheckout">

         <value>false</value>

     </property>

 </bean>

[备注2] hibernate.properties数据库连接配置

dataSource.password=123

dataSource.username=root

dataSource.databaseName=test

dataSource.driverClassName=com.[MySQL](http://lib.csdn.net/base/mysql).jdbc.Driver

dataSource.dialect=org.hibernate.dialect.MySQL5Dialect

dataSource.serverName=localhost:3306

dataSource.url=jdbc:mysql://localhost:3306/test

dataSource.properties=user=dataSource.username;databaseName={dataSource.databaseName};serverName=dataSource.serverName;password={dataSource.password}

dataSource.hbm2ddl.auto=update