配置过程 （springmvc 和 struts）

Spring

Springmvc 的一些简单的配置 和入门使用

一般需要支持包 在spring的required 都需要

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

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

在web.xml 的配置

<!-- 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 <a href="http://www.springframework.org/schema/context/spring-context-3.0.xsd">http://www.springframework.org/schema/context/spring-context-3.0.xsd</a>">

<!-- 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>

关于视图解析 和跳转的处理

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

关于control层

Springmvc 的详细教程 在 <http://elim.iteye.com/blog/1753271>

一个类的被标记@control 都是control层

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";

}

}

Shiro 配置

首先是在web.xml 中

<filter>

<filter-name>shiroFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

<init-param>

<param-name>targetFilterLifecycle</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>shiroFilter</filter-name>

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

</filter-mapping>

然后在applicationContext.xml 文件

配置cachemanager

<bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">

<!-- Set a net.sf.ehcache.CacheManager instance here if you already have one. If not, a new one

will be creaed with a default config:

<property name="cacheManager" ref="ehCacheManager"/> -->

<!-If you don't have a pre-built net.sf.ehcache.CacheManager instance to inject, but you want

a specific Ehcache configuration to be used, specify that here. If you don't, a default

will be used.:

<property name="cacheManagerConfigFile" value="classpath:some/path/to/ehcache.xml"/> -->

</bean>

Eacahe jar包在hibernate中可以找到 然后配置一个eacahe.xml 文件

配置realm （实现了realm的接口）

<bean id="jdbcRealm" class="org.apache.shiro.samples.spring.realm.SaltAwareJdbcRealm">

<property name="name" value="jdbcRealm"/>

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

<property name="credentialsMatcher">

<!-- The 'bootstrapDataPopulator' Sha256 hashes the password

(using the username as the salt) then base64 encodes it: -->

<bean class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">

<property name="hashAlgorithmName" value="SHA-256"/>

<!-- true means hex encoded, false means base64 encoded -->

<property name="storedCredentialsHexEncoded" value="false"/>

</bean>

</property>

</bean>

配置lifecycleBeanPostProcessor

<bean id="lifecycleBeanPostProcessor" class="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>

必须在开启spring 的注解过程中才可以使用

<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"

depends-on="lifecycleBeanPostProcessor"/>

<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">

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

</bean>

<bean id="secureRemoteInvocationExecutor" class="org.apache.shiro.spring.remoting.SecureRemoteInvocationExecutor">

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

</bean>

配置shiroFilter (id必须和配置文件中的)

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

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

<property name="loginUrl" value="/s/login"/>

<property name="successUrl" value="/s/index"/>

<property name="unauthorizedUrl" value="/s/unauthorized"/>

资源是否可以被匿名访问

Authc 必须是在认证之后才可以访问

<property name="filterChainDefinitions">

<value>

/favicon.ico = anon

/logo.png = anon

/shiro.css = anon

/s/login = anon

# allow WebStart to pull the jars for the swing app:

/\*.jar = anon

# everything else requires authentication:

/\*\* = authc

</value>

</property>

</bean>

Realm

LifecycleBeanPost

shiroFilter （和web.xml一致）

Shiro 的认证过程

Url拦截 有优先级，先匹配的可以 覆盖后面的配置

（outline 查看大纲）

认证过程

使用realm获取安全数据

1 获取当前的subject 调用SecurityUtils.getSubject();

2 测试当前的用户是否已经被认证， 调用subject的isAutjenticated()

3 若没有认证，则把用户密码和账号封装为 UsernamePasswordToken 对象

1. 创建一个表单页
2. 把请求提交到Springmvc的handle 中
3. 获取用户账号和密码

4执行登录：调用subject的login(AuthenticationToken)

5 自定义realm方法。从数据库中获取对应的记录 返回给shiro

6 由shiro完成对密码的对比