1. 前端视图层使用bootstrap框架
2. Java代码层使用springmvc mybatis mysql 使用springboot进行整合
3. springboot整合代码。把配置文件转为配置类，使用注解实现。@Configuration 配置文件和 @Bean配置类。@PropertySource属性文件，适应@value来取。

3.1、1、设置springboot的parent

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.2.RELEASE</version>

</parent>

3.2、导入spring boot的web支持

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

3.3、导入外置tomcat，支持jsp依赖

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

3.4配置mybatis框架融合依赖

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>1.3.0</version>

</dependency>

3.5配置数据源，用的德鲁伊

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.18</version>

</dependency>

1. 配置类

4.1 配置数据源bean

@Bean

@Primary

@ConfigurationProperties(prefix="spring.datasource")

public DataSource dataSource(){

return DataSourceBuilder.create().type(DruidDataSource.class).build();

}

4.2配置sqlsession bean

@Bean

**public** SqlSessionFactoryBean sqlSessionFactory(DataSource dataSource){

SqlSessionFactoryBean ssf=**new** SqlSessionFactoryBean();

ssf.setDataSource(dataSource);

ResourcePatternResolver resolver = **new** PathMatchingResourcePatternResolver();

**try** {

ssf.setMapperLocations(resolver.getResources("classpath:mapping/\*.xml"));

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

ssf.setTypeAliasesPackage("com.entity");

**return** ssf;

}

4.3配置dao的实现层bean

@Bean

**public** MapperScannerConfigurer mapperScannerConfigurer(){

MapperScannerConfigurer msc=**new** MapperScannerConfigurer();

msc.setBasePackage("com.dao");

//msc.setSqlSessionFactoryBeanName("sqlSessionFactoryName");

**return** msc;

}

4.4该配置为德鲁伊连接池的映射文件，其属性需要设置的较多具体如下application.properties：

##Druid##

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/oasubject?useUnicode=true&amp;characterEncoding=UTF-8

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.initialSize=5

spring.datasource.minIdle=5

spring.datasource.maxActive=20

spring.datasource.maxWait=60000

spring.datasource.timeBetweenEvictionRunsMillis=60000

spring.datasource.minEvictableIdleTimeMillis=300000

spring.datasource.validationQuery=SELECT 1 FROM DUAL

spring.datasource.testWhileIdle=true

spring.datasource.testOnBorrow=false

spring.datasource.testOnReturn=false

spring.datasource.poolPreparedStatements=true

spring.datasource.maxPoolPreparedStatementPerConnectionSize=20

spring.datasource.filters=stat,wall,log4j

spring.datasource.connectionProperties=druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000

spring.datasource.useGlobalDataSourceStat=true

推荐使用c3p0

导入依赖 </dependency>

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

</dependency>

4.1 配置数据源bean改为如下：

@Bean(destroyMethod="close")

**public** DataSource dataSource(){

ComboPooledDataSource cpd = **new** ComboPooledDataSource();

**try** {

cpd.setDriverClass(driverName);

cpd.setJdbcUrl(url);

cpd.setUser(name);

cpd.setPassword(password);

} **catch** (PropertyVetoException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** cpd;

}

1. 整合shiro框架

5.1、导入依赖jar包

|  |  |
| --- | --- |
| **核心包shiro-core** | **1.2.0** |
| Web相关包shiro-web | 1.2.0 |
| 缓存包shiro-ehcache | 1.2.0 |
| 与spring整合包shiro-spring | 1.2.0 |
| Ehcache缓存核心包ehcache-core | 2.5.3 |
| Shiro自身日志包slf4j-jdk14 | 1.6.4 |

5.2shirorealm登陆和授权

/\*\*

\* shiro realm 认证授权

\* **@author** Administrator

\*

\*/

**public** **class** ShiroRealm **extends** AuthorizingRealm {

//使用类型注解匹配调取业务接口

@Autowired

**private** LoginService ls;

@Autowired

**private** User\_roleService urs;

@Autowired

**private** RoleService rs;

//授权

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// **TODO** Auto-generated method stub

String usercode= (String) principals.getPrimaryPrincipal();

Login login=ls.login(usercode);

Integer uid= login.getuId();

List<String> roles=**null**;

List<User\_role> userRoles=urs.getUserRoles(uid);

**for** (User\_role user\_role : userRoles) {

Role role= rs.getRoles(user\_role.getrId());

roles.add(role.getrName());

}

SimpleAuthorizationInfo info=**new** SimpleAuthorizationInfo();

info.addStringPermissions(roles);

**return** info;

}

//登陆 认证

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {

// **TODO** Auto-generated method stub

UsernamePasswordToken utoken=(UsernamePasswordToken) token;

String usercode=utoken.getUsername();

Login login=ls.login(usercode);

ByteSource salt=ByteSource.Util.*bytes*(usercode);//加密盐

//使用现实方法创建对象，构造方法参数使用（用户名、密码、加密盐、当前方法名）

AuthenticationInfo authenticationInfo =**new** SimpleAuthenticationInfo(usercode,login.getLoginPassword(),salt,**this**.getName());

**return** authenticationInfo;

}

}

5.3shiro配置

/\*\*

\* shiroFilter 过滤器练配置

\*/

@Bean

**public** ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager){

ShiroFilterFactoryBean sff=**new** ShiroFilterFactoryBean();

sff.setSecurityManager(securityManager);//加载安全管理器

sff.setLoginUrl("/login");//登陆url

sff.setSuccessUrl("/home");//登陆成功url

//创建过滤器练的map参数对象

LinkedHashMap<String, String> filterChainDefinitionMap=**new** LinkedHashMap<>();

filterChainDefinitionMap.put("/jsp/login.jsp\*", "anon"); //表示可以匿名访问

filterChainDefinitionMap.put("/loginUser", "anon");

filterChainDefinitionMap.put("/logout\*","anon");

filterChainDefinitionMap.put("/jsp/error.jsp\*","anon");

filterChainDefinitionMap.put("/jsp/index.jsp\*","authc");

filterChainDefinitionMap.put("/\*", "authc");//表示需要认证才可以访问

filterChainDefinitionMap.put("/\*\*", "authc");//表示需要认证才可以访问

filterChainDefinitionMap.put("/\*.\*", "authc");

//配置访问权限，过滤器练设置

sff.setFilterChainDefinitionMap(filterChainDefinitionMap);

**return** sff;

}

/\*\*

\*配置安全管理器

\*/

@Bean

**public** SecurityManager securityManager(Realm realm,CacheManager cacheManager){

DefaultWebSecurityManager defaultWebSecurityManager=**new** DefaultWebSecurityManager();

defaultWebSecurityManager.setRealm(realm);//设置域，桥梁属性

defaultWebSecurityManager.setCacheManager(cacheManager);//缓存设置

**return** defaultWebSecurityManager;

}

/\*\*

\* 缓存设置

\* **@return**

\*/

@Bean

**public** EhCacheManager ehCacheManager(){

EhCacheManager cacheManager = **new** EhCacheManager();

cacheManager.setCacheManagerConfigFile("classpath:config/ehcache-shiro.xml");

**return** cacheManager;

}

/\*\*

\* realm設置

\*/

@Bean

**public** Realm realm(CredentialsMatcher credentialsMatcher){

ShiroRealm shiroRealm= **new** ShiroRealm();

shiroRealm.setCredentialsMatcher(credentialsMatcher);

**return** shiroRealm;

}

/\*\*

\* 加密設置

\* **@return**

\*/

@Bean

**public** CredentialsMatcher credentialsMatcher(){

HashedCredentialsMatcher hashedCredentialsMatcher=**new** HashedCredentialsMatcher();

hashedCredentialsMatcher.setHashAlgorithmName("MD5");

hashedCredentialsMatcher.~~setHashSalted~~(**true**);

**return** hashedCredentialsMatcher;

}

/\*方法授權驗證\*/

5.4 在resources下配置application.properties文件，配置jsp路径

spring.mvc.view.prefix=/WEB-INF/view/

spring.mvc.view.suffix=.jsp