**1、本章面试题**

       springboot+mybatis 整合 shiro

**2、知识点**

**2.1、课程回顾**

               什么是springboot,它的特点

springboot整合mybatis

**2.2、本章重点**

         springboot整合连接池 druid(c3p0,dbcp,proxool)

springboot通用日志 配置

springboot整合shiro(重点)

**3、具体内容**

**3.1 整合连接池**

**添加支持**

     <!-- druid数据库连接池 -->

     <dependency>

      <groupId>com.alibaba</groupId>

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

      <version>1.1.9</version>

     </dependency>

**修改配置**

#spring.datasource.druid.driver-class-name=oracle.jdbc.driver.OracleDriver  可配可不配，阿里的数据库连接池会通过url自动搜寻

spring.datasource.druid.url=jdbc:oracle:thin:@localhost:1521:orcl

spring.datasource.druid.username=scott

spring.datasource.druid.password=tiger

spring.datasource.druid.initial-size=5

spring.datasource.druid.max-active=20

spring.datasource.druid.min-idle=10

spring.datasource.druid.max-wait=10

#是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的数据库性能提升巨大，比如说oracle。

#在mysql5.5以下的版本中没有PSCache功能，建议关闭掉。

#spring.datasource.druid.pool-prepared-statements=true

#配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒

spring.datasource.druid.time-between-eviction-runs-millis=60000

#配置一个连接在池中最小生存的时间，单位是毫秒

spring.datasource.druid.min-evictable-idle-time-millis=300000

#配置扩展插件:监控统计用的filter:stat  日志用的filter:log4j  防御sql注入的filter:wall

#spring.datasource.druid.filters=stat,wall

#spring.datasource.druid.filter.stat.log-slow-sql=true

#spring.datasource.druid.filter.stat.slow-sql-millis=2000

**3.2 springboot日志配置：**

**配置日志不需要导入jar，日志包在父项目中已经引入，具体位置如下：**

spring-boot-starter-parent ----父包---> spring-boot-dependencies ----父包--->spring-boot-dependencies -----查找----> spring-boot-starter-logging ---点击进去----> 能看到日志包：logback-classic，log4j-to-slf4j.....

**在resources下创建logback.xml或者logback-spring.xml，复制配置进去**

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

<configuration debug="false">

    <!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径-->

    <property name="LOG\_HOME" value="D:/projects/log" />

    <!-- 控制台输出 -->

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">

            <!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->

            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>

        </encoder>

    </appender>

    <!-- 按照每天生成日志文件 -->

    <appender name="FILE"  class="ch.qos.logback.core.rolling.RollingFileAppender">

        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

            <!--日志文件输出的文件名-->

            <FileNamePattern>${LOG\_HOME}/springbootdemo.log.%d{yyyy-MM-dd}.log</FileNamePattern>

            <!--日志文件保留天数-->

            <MaxHistory>30</MaxHistory>

        </rollingPolicy>

        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">

            <!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->

            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>

        </encoder>

        <!--日志文件最大的大小-->

        <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">

            <MaxFileSize>10MB</MaxFileSize>

        </triggeringPolicy>

    </appender>

    <!--myibatis log configure-->

    <logger name="com.apache.ibatis" level="DEBUG"/>

    <logger name="java.sql.Connection" level="DEBUG"/>

    <logger name="java.sql.Statement" level="DEBUG"/>

    <logger name="java.sql.PreparedStatement" level="DEBUG"/>

    <!-- 日志输出级别

      trace<debug<info<warn<error<fatal

    级别之间是包含的关系，意思是如果你设置日志级别是trace，则大于等于这个级别的日志都会输出。

    trace： 是追踪，就是程序推进以下，你就可以写个trace输出，所以trace应该会特别多，不过没关系，我们可以设置最低日志级别不让他输出。

    debug： 调试么，我一般就只用这个作为最低级别，trace压根不用。是在没办法就用eclipse或者idea的debug功能就好了么。

    info： 输出一下你感兴趣的或者重要的信息，这个用的最多了。

    warn： 有些信息不是错误信息，但是也要给程序员的一些提示，类似于eclipse中代码的验证不是有error 和warn（不算错误但是也请注意，比如以下depressed的方法）。

    error： 错误信息。用的也比较多。

    fatal： 级别比较高了。重大错误，这种级别你可以直接停止程序了，是不应该出现的错误么！不用那么紧张，其实就是一个程度的问题。

   -->

    <root level="DEBUG">

        <appender-ref ref="STDOUT" />

        <appender-ref ref="FILE" />

    </root>

    <!--日志异步到数据库 -->

    <!--<appender name="DB" class="ch.qos.logback.classic.db.DBAppender">-->

    <!--<!–日志异步到数据库 –>-->

    <!--<connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">-->

    <!--<!–连接池 –>-->

    <!--<dataSource class="com.mchange.v2.c3p0.ComboPooledDataSource">-->

    <!--<driverClass>com.mysql.jdbc.Driver</driverClass>-->

    <!--<url>jdbc:mysql://127.0.0.1:3306/databaseName</url>-->

    <!--<user>root</user>-->

    <!--<password>root</password>-->

    <!--</dataSource>-->

    <!--</connectionSource>-->

    <!--</appender>-->

</configuration>

**3.3 springboot整合shiro**

**springboot 的application.properties配置**

<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

**springboot访问templates配置(或者创建resources放置HTML)**

spring.resources.static-locations=classpath:/templates/

**整合前需要先讲解讲解@Configuration和@Bean用法，否则整合代码看不懂。**

@Configuration 可理解为用spring的时候xml里面的<beans>标签，注解标记在类上,就像之前我们声明的一个spring的xml配置文件,该类我们称为配置类.

@Bean 可理解为用spring的时候xml里面的<bean>标签，标记在方法之上,方法的返回值为向springIOC容器之中注入一个Bean.其中,返回值相当于xml文件bean标签的class属性.如果@Bean配置value相当于id属性，如果没有配置,方法的名称相当于id属性

package com.aaa.sbms.config;

import com.aaa.sbms.entity.Dept;

import com.aaa.sbms.entity.Emp;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

/\*\*

\* fileName:MyConfig

\* description:

\* author:zz

\* createTime:2019/11/26 15:48

\* version:1.0.0

\*/

@Configuration

public class MyConfig {

/\*\*

\* 相当于过去的

\* <bean name="emp" class="com.aaa.sbms.entity.Emp" >

\* @return

\*/

@Bean

public Emp emp(@Qualifier("d1") Dept d){

Emp emp = new Emp();

emp.setEname("zhangsan");

emp.setSal(10000);

emp.setDept(d);

return emp;

}

@Bean("d1")

public Dept dept(){

Dept dept =new Dept();

dept.setDname("kaifa1");

dept.setLoc("1lou");

return dept;

}

}

package com.aaa.sbms;

import com.aaa.sbms.entity.Emp;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.transaction.annotation.EnableTransactionManagement;

@SpringBootApplication

@MapperScan("com.aaa.sbms.dao")

@EnableTransactionManagement //开启注解式事务

public class SbmShiroApplication {

public static void main(String[] args) {

ConfigurableApplicationContext applicationContext = SpringApplication.run(SbmShiroApplication.class, args);

Emp emp =(Emp)applicationContext.getBean("emp");

System.out.println(emp.getEname()+","+emp.getSal());

System.out.println(emp.getDept().getDname()+","+emp.getDept().getLoc());

}

}

上面知识点熟悉后，整合shiro

**1，引入包：**

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.4.0</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-ehcache</artifactId>

<version>1.4.0</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.4.0</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>1.4.0</version>

</dependency>

**2,编写ShiroConfig**

package com.aaa.sm.config;

import org.apache.shiro.mgt.SecurityManager;

import org.apache.shiro.spring.LifecycleBeanPostProcessor;

import org.apache.shiro.spring.web.ShiroFilterFactoryBean;

import org.apache.shiro.web.mgt.DefaultWebSecurityManager;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.util.LinkedHashMap;

import java.util.Map;

/\*\*

\* fileName:ShiroConfiguration

\* description:

\* author:zz

\* createTime:2019/11/14 10:32

\* version:1.0.0

\*/

@Configuration

public class ShiroConfiguration {

/\*\*

\* 拦截 跳转配置

\* @param securityManager

\* @return

\*/

@Bean

public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager){

ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();

//设值

shiroFilterFactoryBean.setSecurityManager(securityManager);

//未认证跳转登录页面

shiroFilterFactoryBean.setLoginUrl("/html/login.html");

//认证成功后跳转的页面

//shiroFilterFactoryBean.setSuccessUrl("/");

//为授权跳转页面

//shiroFilterFactoryBean.setUnauthorizedUrl("/");

//拦截所有的/\*\*= authc 的配置必须在最下面，所以使用LinkedHashMap

Map chainDefinitionMap =new LinkedHashMap();

//放开登录相关

chainDefinitionMap.put("/html/login.html","anon");

chainDefinitionMap.put("/user/login","anon");

//放开静态资源

chainDefinitionMap.put("/css/\*\*","anon");

chainDefinitionMap.put("/js/\*\*","anon");

chainDefinitionMap.put("/imgs/\*\*","anon");

//拦截除上面放开之外的所有请求

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

shiroFilterFactoryBean.setFilterChainDefinitionMap(chainDefinitionMap);

return shiroFilterFactoryBean;

}

/\*\*

\* 把shiro的bean交给spring管理

\* @return

\*/

@Bean

public LifecycleBeanPostProcessor lifecycleBeanPostProcessor(){

return new LifecycleBeanPostProcessor();

}

/\*\*

\* SecurityManager 核心配置

\* @param myRealm

\* @return

\*/

@Bean

public SecurityManager securityManager(MyShiroRealm myRealm){

DefaultWebSecurityManager defaultWebSecurityManager = new DefaultWebSecurityManager();

defaultWebSecurityManager.setRealm(myRealm);

return defaultWebSecurityManager;

}

/\*\*

\* 自定义realm配置

\* @return

\*/

@Bean

public MyShiroRealm myRealm(){

return new MyShiroRealm();

}

}

**3.myrealm**

package com.aaa.sm.config;

import com.aaa.sm.service.UserService;

import org.apache.shiro.SecurityUtils;

import org.apache.shiro.authc.\*;

import org.apache.shiro.authz.AuthorizationInfo;

import org.apache.shiro.authz.SimpleAuthorizationInfo;

import org.apache.shiro.realm.AuthorizingRealm;

import org.apache.shiro.subject.PrincipalCollection;

import org.springframework.beans.factory.annotation.Autowired;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

/\*\*

\* fileName:MyShiroRealm

\* description:

\* author:zz

\* createTime:2019/11/14 10:38

\* version:1.0.0

\*/

public class MyShiroRealm extends AuthorizingRealm {

@Autowired

private UserService userService;

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {

return null;

}

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

//获取用户名

Object userName = authenticationToken.getPrincipal();

Map map = new HashMap();

map.put("userName",userName);

//根据用户名查询列表

List<Map> userMapList = userService.getListByParam(map);

//判断

if(userMapList!=null&&userMapList.size()>0){

Map userMap = userMapList.get(0);

//SecurityUtils.getSubject().getSession().setAttribute("userInfo",userMap);

return new SimpleAuthenticationInfo(userMap.get("USER\_NAME")+"",

userMap.get("PASSWORD")+"",getName());

}else {

throw new AccountException();

}

}

}

**4,编写登录页面和功能，进行测试**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>用户登录</title>

<script src="/js/jquery-3.4.1.min.js"></script>

<script>

$(function () {

//绑定登录按钮点击事件

$("#btn").click(function () {

$.ajax({

url:"/user/login",

data:{userName:$("#userN").val(),passWord:$("#passW").val()},

type:"post",

dataType:"json",

success:function (data) {

alert(data);

alert(data=='1');

if(data=='1'){

alert(222);

//跳转首页

location.href="/html/index.html";

}else{

$("#errorDiv").html("用户名或者密码错误！");

}

},

error:function () {

alert("error");

}

});

});

});

</script>

</head>

<body>

<div style="color: red;" id="errorDiv"></div>

用户名：<input type="text" id="userN">

密码：<input type="text" id="passW">

<input type="button" id="btn" value="登录">

</body>

</html>

**3.4mybatis分页插件（非官网）**

**1，添加jar包**

  <!-- 分页插件pagehelper -->

  <dependency>

 <groupId>com.github.pagehelper</groupId>

      <artifactId>pagehelper</artifactId>

        <version>5.0.0</version>

</dependency>

 <dependency>

 <groupId>com.github.pagehelper</groupId>

       <artifactId>pagehelper-spring-boot-autoconfigure</artifactId>            <version>1.2.3</version>

    </dependency>        <dependency>

          <groupId>com.github.pagehelper</groupId>

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

           <version>1.2.3</version>

 </dependency>

 <!-- 分页插件pagehelper -->

**2，添加springboot配置**

[**https://pagehelper.github.io/**](https://pagehelper.github.io/)

[**https://github.com/pagehelper/Mybatis-PageHelper/blob/master/README\_zh.md**](https://github.com/pagehelper/Mybatis-PageHelper/blob/master/README_zh.md)

 #分页插件

#helperDialect属性来指定分页插件使用哪种方言

pagehelper.helper-dialect=oracle

#当该参数设置为 true 时，pageNum<=0 时会查询第一页， pageNum>pages（超过总数时），会查询最后一页。

pagehelper.reasonable=true

#支持通过 Mapper 接口参数来传递分页参数，默认值false，分页插件会从查询方法的参数值中，自动根据上面 params 配置的字段中取值，查找到合适的值时就会自动分页。

pagehelper.support-methods-arguments=true

#增加了该参数来配置参数映射，用于从对象中根据属性名取值

pagehelper.params=count=countSql

**3，具体使用**

   与以前controller区别

  //设置当前第几页和每页显示数量    PageHelper.startPage(Integer.valueOf(map.get("pageNo")+""),Integer.valueOf(map.get("pageSize")+""));

        //用PageInfo对结果进行包装

        PageInfo<Map> pageInfo =new PageInfo<Map>(newsService.getList());

/\*\*

     \* 分页部门查询

     \* @param map

     \* @return

     \*/

    @ResponseBody

    @RequestMapping("page")

    public Object page(@RequestParam Map map){

        int pageNo = Integer.valueOf(map.get("pageNo")+"");

        int pageSize = Integer.valueOf(map.get("pageSize")+"");

        //初始化配置

        PageHelper.offsetPage(pageNo,pageSize);

        PageInfo<Map> pageInfo = new PageInfo<Map>(deptService.getList());

       //如果使用easyui可以这样封装,其他框架，自己根据pageInfo解析

        Map tmap  = new HashMap();

        tmap.put("total",pageInfo.getTotal());

        tmap.put("rows",pageInfo.getList());

        return tmap;

    }

**3.5 springboot 启动执行sql语句**

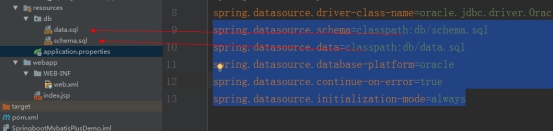
spring.datasource.schema=classpath:db/schema.sql

spring.datasource.data=classpath:db/data.sql

spring.datasource.database-platform=oracle

spring.datasource.continue-on-error=true

spring.datasource.initialization-mode=always



**4、本章总结**

**4.1 总结本章知识点**

**4.2 面试题答案**

**4.3 预习下一章重点**