1. springBoot配置tomcat

spring-boot默认提供内嵌的tomcat，所以打包直接生成jar包，用java -jar命令就可以启动。但是，有时候我们更希望一个tomcat来管理多个项目，这种情况下就需要项目是war格式的包而不是jar格式的包。

1.1、将项目的启动类Application.java继承SpringBootServletInitializer并重写configure方法

@SpringBootApplication

@ImportResource(locations = "classpath\*:spring/\*.xml")

public class Application extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

return application.sources(Application.class);

}

public static void main(String[] args) throws Exception {

SpringApplication.run(Application.class, args);

}

}

1.2、在pom.xml文件中，project下面增加package标签为war

<groupId>com.test.lsy</groupId>

<artifactId>pine</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

1.3、还是在pom.xml文件中，dependencies下面添加

<dependency>

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

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

<scope>provided</scope>

</dependency>

1.4、如果需要在springboot中加上request前缀，需要在application.properties中添加server.contextPath=/prefix/即可。其中prefix为前缀名。这个前缀会在war包中失效，取而代之的是war包名称，如果war包名称和prefix相同的话，那么调试环境和正式部署环境就是一个request地址了。

1. springBoot配置thymeleaf

2.1、在pom.xml加入依赖

<dependency>

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

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

</dependency>

2.2、在application.properties进行配置

# THYMELEAF (ThymeleafAutoConfiguration)

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html

spring.thymeleaf.mode=HTML5

spring.thymeleaf.encoding=UTF-8

spring.thymeleaf.content-type=text/html

spring.thymeleaf.cache=false

2.3建立一个文件src/main/resouces/spring/spring.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:mvc="http://www.springframework.org/schema/mvc"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">  
  
 <context:component-scan base-package="com.lsy.mybatis.test"></context:component-scan>  
  
</beans>

2.4、下面是测试一下

编写模板文件src/main/resouces/templates/index.html

Controller映射到页面test/index：

@ResponseBody

@RequestMapping(value = "index",method = RequestMethod.GET)

public ModelAndView access(){

ModelAndView modelAndView = new ModelAndView("index");

return modelAndView;

}

1. 整合myBatis和数据源

3.1、添加依赖

<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>1.1.1</version>  
</dependency>  
<dependency>  
 <groupId>tk.mybatis</groupId>  
 <artifactId>mapper</artifactId>  
 <version>3.3.6</version>  
</dependency>

<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.0.16</version>  
</dependency>

<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
</dependency>

3.2、在application.properties进行配置

spring.datasource.url=jdbc:mysql://localhost:3306/ssm\_shiro?useUnicode=true&characterEncoding=utf-8  
spring.datasource.username=root  
spring.datasource.password=123456  
spring.datasource.driverClassName=com.mysql.jdbc.Driver  
spring.datasource.maxActive=10  
spring.datasource.maxWait=10000  
spring.datasource.minIdle=5  
spring.datasource.initialSize=5

# 使用druid数据源  
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  
spring.datasource.filters= stat  
spring.datasource.timeBetweenEvictionRunsMillis= 60000  
spring.datasource.minEvictableIdleTimeMillis= 300000  
spring.datasource.validationQuery=select 'x'  
spring.datasource.testWhileIdle=true  
spring.datasource.testOnBorrow=false  
spring.datasource.testOnReturn=false  
spring.datasource.poolPreparedStatements=true  
spring.datasource.maxOpenPreparedStatements=20

3.3编写一个MybatisConfig 配置类

@Configuration  
@ConditionalOnBean(value = DataSource.class)  
public class MybatisConfig {  
 @Autowired  
 DataSource dataSource;  
 @Bean(name = "sqlSessionFactory")  
 public SqlSessionFactory sqlSessionFactoryBean() {  
 SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  
 bean.setDataSource(dataSource);  
 bean.setTypeAliasesPackage("com.lsy.mybatis.test.model");  
 try {  
 SqlSessionFactory factory = bean.getObject();  
 factory.getConfiguration().setMapUnderscoreToCamelCase(true);  
 factory.getConfiguration().setCacheEnabled(true);  
 return factory;  
 } catch (Exception e) {  
 e.printStackTrace();  
 throw new RuntimeException(e);  
 }  
 }

@Bean  
 public SqlSessionTemplate sqlSessionTemplate(SqlSessionFactory sqlSessionFactory) {  
 return new SqlSessionTemplate(sqlSessionFactory);  
 }  
}

3.3扫描mapper配置

@Configuration

@AutoConfigureAfter(MybatisConfig.class)//在指定的配置类初始化后再加载   
public class MyBatisMapperScannerConfig{  
 @Bean//Mapper扫描配置  
 public MapperScannerConfigurer mapperScannerConfigurer() {  
 StringBuilder mapperPackageBuilder = new StringBuilder();  
 mapperPackageBuilder.append("com.lsy.mybatis.test.mapper");  
 MapperScannerConfigurer mapperScannerConfigurer = new MapperScannerConfigurer();  
 mapperScannerConfigurer.setSqlSessionFactoryBeanName("sqlSessionFactory");  
 mapperScannerConfigurer.setBasePackage(mapperPackageBuilder.toString());  
 Properties properties = new Properties();  
 properties.setProperty("notEmpty", "false");  
 properties.setProperty("IDENTITY", "MYSQL");  
 mapperScannerConfigurer.setProperties(properties);  
 return mapperScannerConfigurer;  
 }  
}