**SpringBoot**

# 什么是SpringBoot?

SpringBoot是一个微服务框架-分布式，用来简化我们的XML配置，提高开发效率。

**特点：**

1.嵌入Tomcat，无需部署war文件

2.简化Maven配置

3.自动配置bean

4.开箱即用,无需配置xml

# 2. 创建SpringBoot项目

## 1. 创建一个Maven项目

## 2. 引入springboot必须依赖.

<**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>1.3.3.RELEASE</**version**>  
</**parent**>

## 3. 添加springboot-web组件

<**dependencies**>  
 *<!-- 引入SpringBoot Web 组件 -->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
</**dependencies**>

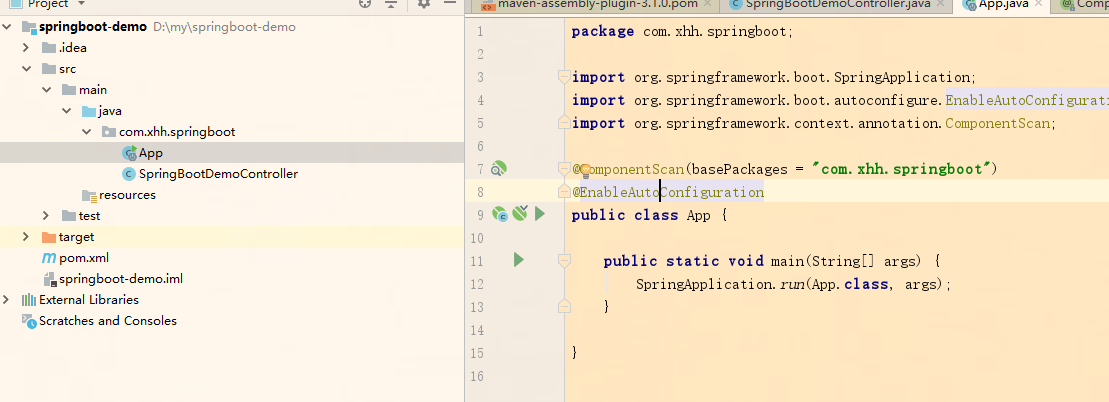
**4．编写Controller**

@RestController  
@EnableAutoConfiguration  
**public class** SpringBootDemoController {  
  
  
 @RequestMapping(**"/hello"**)  
 **public** String hello(@RequestParam String name){  
 **return** name + **"， 你好"**;  
 }  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(SpringBootDemoController.**class**, args);  
 }  
  
}

直接运行main方法即可。

**5. 将启动入口单独放置**

@ComponentScan(basePackages = **"com.xhh.springboot"**)  
@EnableAutoConfiguration  
**public class** App {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(App.**class**, args);  
 }  
  
}



# 3. SpringBoot静态图片访问

在我们开发web应用的时候，经常会有大量的js, css图片等静态文件资源。

默认配置：

SpringBoot默认提供静态资源目录需要置于classpath路径下，目录名需要符合规则：

/static

/public

/resources

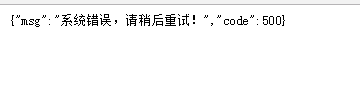
/META-INF/resources

举例：

我们可以在src/main/resources/目录下创建static文件目录,在目录下面放置一张D.jpg图片。启动程序后，尝试访问<http://localhost:8080/D.jpg>。则证明配置成功。

# 4. SpringBoot全局异常处理

@ControllerAdvice  
**public class** GlobalExceptionHandler {  
  
 @ExceptionHandler(RuntimeException.**class**)  
 @ResponseBody  
 **public** Map<String, Object> exceptionHandle(){  
 Map<String, Object> ret = **new** HashMap<String, Object>();  
 ret.put(**"code"**, 500);  
 ret.put(**"msg"**, **"系统错误，请稍后重试！"**);  
 **return** ret;  
 }  
  
}



**@ControllerAdvice** 是controller的一个辅助注解，最常用的就是做为全局异常处理类

**@ControllerAdvice** 可以指定扫描范围。

**@ControllerAdvice** 返回形式和Controller中一致。

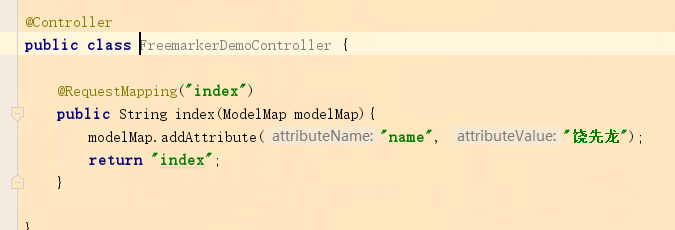
**@ExceptionHandler(Exception.class)** 指定拦截异常类型

# 5. SpringBoot集成Freemarker

## 1. 引入依赖

*<!-- 引入springboot freemarker依赖-->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-freemarker</**artifactId**>  
</**dependency**>

## 后台代码



**在resources文件下创建一个文件夹-templates,创建index.ftl文件**

## 使用freemarker

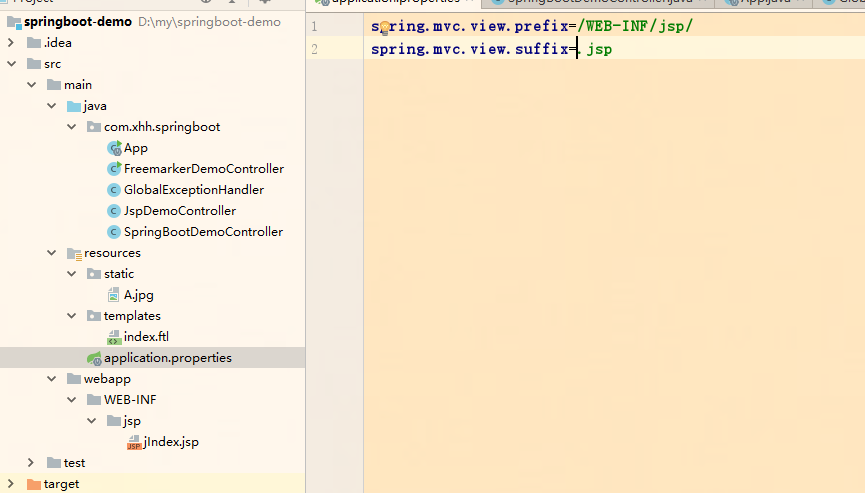
**参考Freemarker开发指南**

# 6. SpringBoot集成JSP

**1. 依赖引入：**

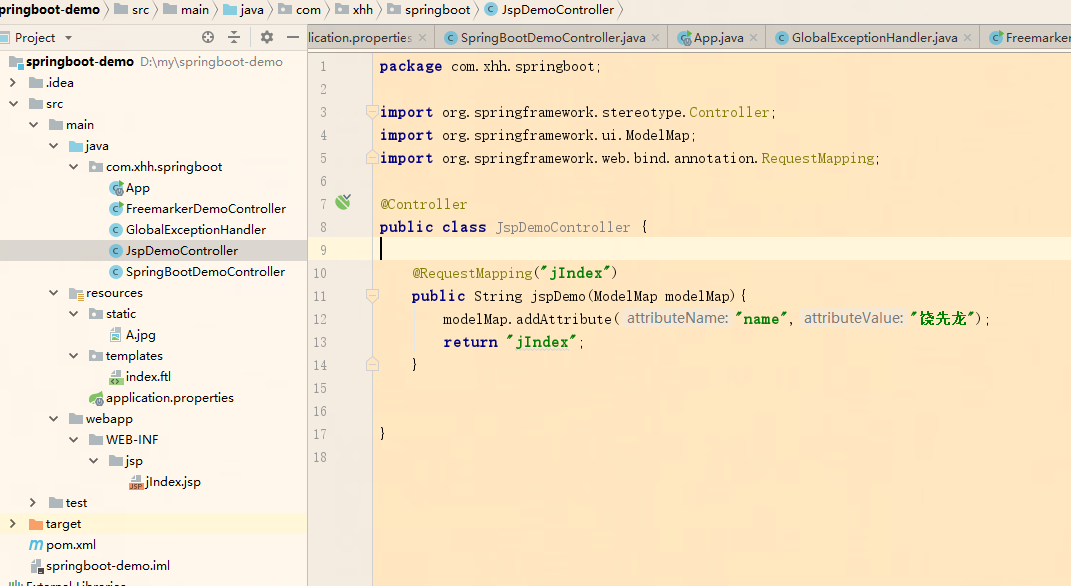
*<!-- 引入JSP依赖 -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-tomcat</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.apache.tomcat.embed</**groupId**>  
 <**artifactId**>tomcat-embed-jasper</**artifactId**>  
</**dependency**>

**2. application.properties文件下添加前缀和后缀：**



**spring.mvc.view.prefix**=**/WEB-INF/jsp/  
spring.mvc.view.suffix**=**.jsp**

**3．编写前后端代码**

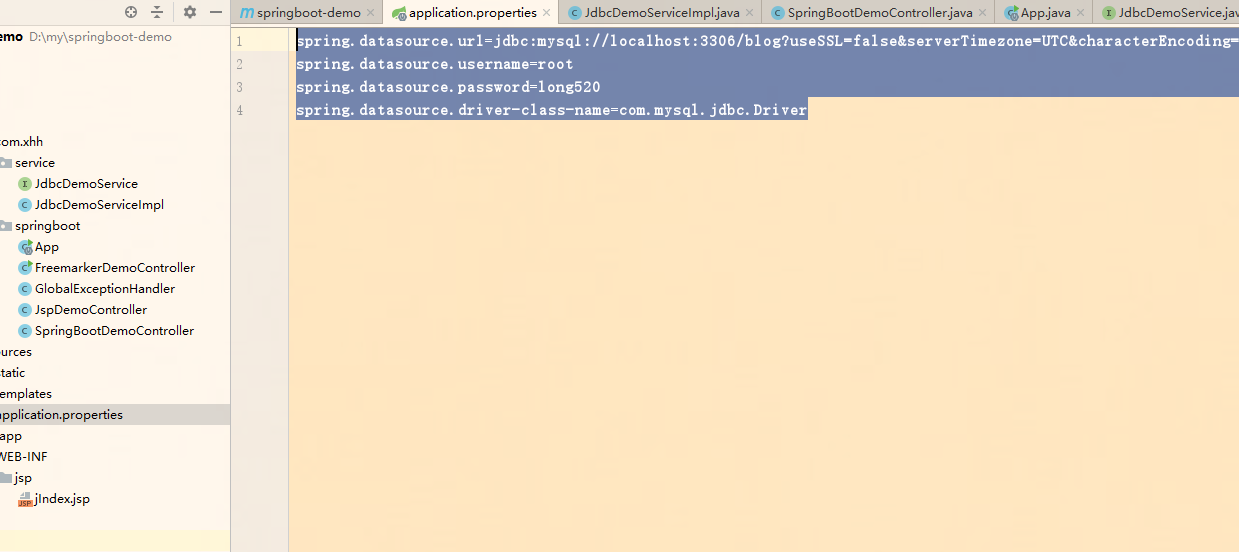


# 7. SpringBoot集成JDBC

**1. 依赖引入：**

*<!-- 引入JDBC依赖 -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-jdbc</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
</**dependency**>

**2.配置数据库连接信息**

**spring.datasource.url**=**jdbc:mysql://localhost:3306/blog?useSSL=false&serverTimezone=UTC&characterEncoding=utf-8  
spring.datasource.username**=**root  
spring.datasource.password**=**long520  
spring.datasource.driver-class-name**=**com.mysql.jdbc.Driver**

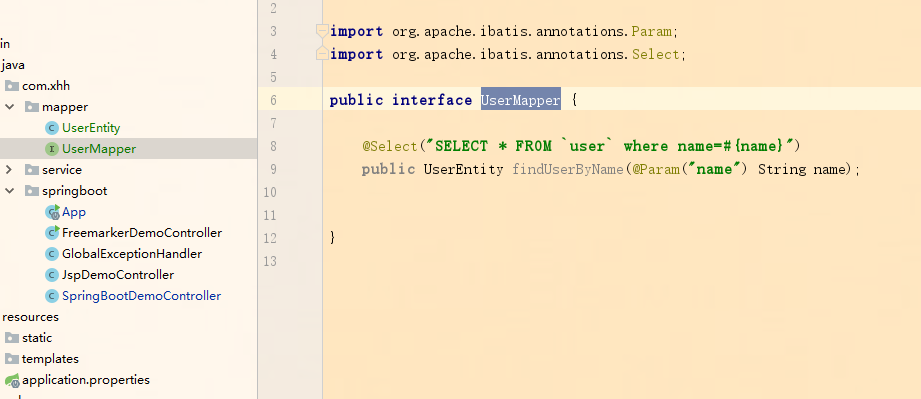
**3.代码**

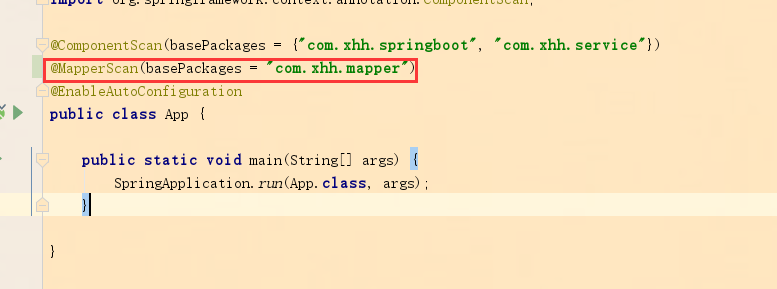


# 8. SpringBoot集成Mybatis

**1. 依赖引入：**

*<!-- 引入mybatis依赖 -->*<**dependency**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 <**version**>1.1.1</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
</**dependency**>

**2.编写mapper**

**3.扫描Mapper**

**4.查询**

# 9. SpringBoot集成JPA

## 1. 引入依赖

*<!-- 引入JPA依赖 -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
</**dependency**>

## 2. 创建实体类映射

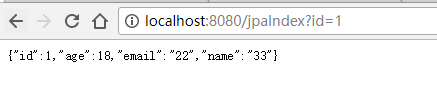
## 3. 创建查询Dao

**package** com.xhh.jpadao;  
  
  
**import** com.xhh.jpaMapper.JPAUserEntity;  
**import** org.springframework.data.jpa.repository.JpaRepository;  
  
**public interface** JPAUserRepository **extends** JpaRepository<JPAUserEntity, Long> {  
  
}

**实现JpaRepository 接口，其子类封装了很多常用公共方法。**

## 4. 扫描

## 5. 测试

@RestController  
**public class** JpaDemoController {  
  
 @Autowired  
 JPAUserRepository **jpaUserRepository**;  
  
 @RequestMapping(**"jpaIndex"**)  
 **public** JPAUserEntity jpaDemo(Long id){  
 **return jpaUserRepository**.findOne(id);  
 }  
  
}

# 10. SpringBoot集成事物

直接在业务逻辑处理方法上加上**@Transactional** 注解

# 11. SpringBoot多数据源(mybatis)

## 1. 配置多个数据库信息

**# 数据源test1：**

**spring.datasource.test1.url**=**jdbc:mysql://localhost:3306/test1?useSSL=false&serverTimezone=UTC&characterEncoding=utf-8  
spring.datasource.test1.username**=**\*\*\*\*\*\*  
spring.datasource.test1.password**=**\*\*\*\*\*\*  
spring.datasource.test1.driver-class-name**=**com.mysql.jdbc.Driver  
  
# 数据源test2：  
spring.datasource.test2.url**=**jdbc:mysql://localhost:3306/test2?useSSL=false&serverTimezone=UTC&characterEncoding=utf-8  
spring.datasource.test2.username**=**\*\*\*\*\*\*  
spring.datasource.test2.password**=**\*\*\*\*\*\*  
spring.datasource.test2.driver-class-name**=**com.mysql.jdbc.Driver**

**如果只有一个数据源,springboot默认会使用spring.datasource作为前缀**

**如：**

**spring.datasource.url**=**jdbc:mysql://localhost:3306/blog?useSSL=false&serverTimezone=UTC&characterEncoding=utf-8  
spring.datasource.username**=**\*\*\*\*\*\*  
spring.datasource.password**=**\*\*\*\*\*\*  
spring.datasource.driver-class-name**=**com.mysql.jdbc.Driver**

## 配置多个数据源

步骤：

1. 配置数据源
2. 配置SqlSessionFactory, 需要数据源作为参数
3. 配置事物管理DataSourceTransactionManager需要数据 源作为参数

**package** com.xhh.datasource;  
  
  
**import** org.apache.ibatis.session.SqlSessionFactory;  
**import** org.mybatis.spring.SqlSessionFactoryBean;  
**import** org.mybatis.spring.SqlSessionTemplate;  
**import** org.mybatis.spring.annotation.MapperScan;  
**import** org.springframework.beans.factory.annotation.Qualifier;  
**import** org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;  
**import** org.springframework.boot.context.properties.ConfigurationProperties;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.context.annotation.Primary;  
**import** org.springframework.jdbc.datasource.DataSourceTransactionManager;  
  
**import** javax.sql.DataSource;  
  
@MapperScan(basePackages = **"com.xhh.mapper1"**,  
 sqlSessionFactoryRef = **"test1SqlSessionFactory"**,  
 sqlSessionTemplateRef = **"test1SqlSessionTemplate"**)  
@Configuration  
**public class** Datasource1 {  
  
 */\*\*  
 \* 配置数据源  
 \** ***@return*** *\*/* @Primary  
 @Bean(name = **"test1Datasource"**)  
 @ConfigurationProperties(prefix = **"spring.datasource.test1"**)  
 **public** DataSource getDataSource(){  
 **return** DataSourceBuilder.*create*().build();  
 }  
  
 */\*\*  
 \* 配置SqlSessionFactory  
 \** ***@param* dataSource** *数据源  
 \** ***@return*** *SqlSessionFactory  
 \** ***@throws*** *Exception  
 \*/* @Primary  
 @Bean(name = **"test1SqlSessionFactory"**)  
 **public** SqlSessionFactory getSqlSessionFactory(@Qualifier(**"test1Datasource"**) DataSource dataSource) **throws** Exception {  
 SqlSessionFactoryBean factoryBean = **new** SqlSessionFactoryBean();  
 factoryBean.setDataSource(dataSource);  
 **return** factoryBean.getObject();  
 }  
  
 */\*\*  
 \* 配置事物管理器  
 \*/* @Primary  
 @Bean(name = **"test1TransactionManager"**)  
 **public** DataSourceTransactionManager getTransactionManager(@Qualifier(**"test1Datasource"**) DataSource dataSource){  
 **return new** DataSourceTransactionManager(dataSource);  
 }  
  
 */\*\*  
 \* 配置SqlSessionTemplate  
 \** ***@return*** *\*/* @Primary  
 @Bean(name = **"test1SqlSessionTemplate"**)  
 **public** SqlSessionTemplate getSqlSessionTemplate(@Qualifier(**"test1SqlSessionFactory"**) SqlSessionFactory sqlSessionFactory){  
 **return new** SqlSessionTemplate(sqlSessionFactory);  
 }  
  
  
}

**注意 ：**

@Primary

这是其中一个数据源的配置信息，除此之外需要配置一个主Bean，在默认情况下使用，

有多个数据库就要配置多少个数据源

通过在数据源中配置的 :

@MapperScan(basePackages = **"com.xhh.mapper1"**,  
 sqlSessionFactoryRef = **"test1SqlSessionFactory"**,  
 sqlSessionTemplateRef = **"test1SqlSessionTemplate"**)

通过扫描不同的包来确定不同的Mapper调用不同的数据库

## 多个数据源测试

@RestController  
**public class** MultiDatasourceController {  
  
 @Autowired  
 DataSourceService1 **service1**;  
 @Autowired  
 DataSourceService2 **service2**;  
 @Autowired  
 MybatisService **service**;  
  
 @RequestMapping(**"add"**)  
 **public** String addUser(){  
 UserEntity user1 = **new** UserEntity(18, **"test1@xhh.com"**, **"test1"**);  
 UserEntity user2 = **new** UserEntity(18, **"test2@xhh.com"**, **"test2"**);  
 UserEntity user3 = **new** UserEntity(18, **"blog@xhh.com"**, **"blog"**);  
 **service1**.add(user1);  
 **service2**.add(user2);  
 **service**.add(user3);  
 **return "success"**;  
 }  
  
}

# 12. SpringBoot分布式事物管理

**使用springboot + jta +atomikos 进行分布式事物管理**