**中软国际考题-笔试**

1. 单选题

1、下列哪个访问端点是SpringBoot actuator提供用于提供应用的基本健康信息（C）

A.info B.metrics C.health D.trace

2、Spring Boot使用一个非常特别的PropertySource次序来允许对值进行合理的覆盖， 其中优先级最高的是：（A）

A. 命令行参数

B. 来自于java:comp/env的JNDI属性

C. Java系统属性（System.getProperties()）

D. 操作系统环境变量

3、spring-boot的全局配置文件有几种写法（B）

A.1 B.2 C.3 D.4

4、使用注解关闭SpringBoot的某个自动配置，使用以下哪个注解（D）

A. @SpringBootConfiguration B. @Configuration

C. @EnableAutoConfiguration D. @SpringBootApplication

5、运行 Spring Boot 有哪几种方式？（B）

A.使用DEBUG运行 B. 打包用命令或者放到容器中运行

C.通过WEB.XML运行 D.通过浏览器运行

6、下列哪个不是SpringBoot数据源的配置项？（D）

A．spring.datasource.url=jdbc:mysql://127.0.0.1:3306/news?useUnicode=true&characterEncoding=utf-8

B. spring.datasource.username=root

C. spring.datasource.password=root

D. spring.datasource.driver-class =com.mysql.jdbc.Drive

7、@SpringBootApplication是一个复合注解，不包含于此注解功能的注解是（B）

A. @ComponentScan B. @SpringBootTest

C. @SpringBootConfiguration D. @EnableAutoConfiguration

8、Spring boot 如何添加开发者工具集(A )?

A: spring-boot-devtools

B: spring-boot-starter

C: spring-starter-ribbon

D: spring-starter-feign

1. 多选题
2. 下面哪些是SpringBoot内置的Servlet容器？（A、B、C）

A.Tomcat B.Jetty C.Undertow D.JBoss

2、使用@SpringBootApplication后等价于以默认属性使用以下哪些注解？（A、B、C）

A. @Configuration B. @EnableAutoConfiguration

C. @ComponentScan D.@Bean

3、Spring Boot 启动连接到内存数据库 H2 的 JPA 应用程序需要哪些依赖项？（B、C、D）

A.redis B.web 启动器 C.h2 D.jpa 数据启动器

4、@SpringBootApplication是一个组合注解，相当于以下哪些注解的组合？（A、D、E）

A. @SpringBootConfiguration B. @Controller C. @Override

D. @EnableAutoConfiguration E . @ComponentScan

5、Spring Boot有哪些优点？（A、B，C，D）

A. 减少开发，测试时间和努力。

B. 使用JavaConfig有助于避免使用XML。

C. 避免大量的Maven导入和各种版本冲突。

D. 通过提供默认值快速开始开发

6、 Spring Boot 的核心配置文件是(A , B)

A. application B. bootstrap C.WEB.XML D.application

7、下列哪些关于SpringBoot的描述是正确的？（ABCE）

A 减少开发，测试时间和努力。

B 使用JavaConfig有助于避免使用XML。

C 避免大量的Maven导入和各种版本冲突。

D 提供多语言支持。

E 通过提供默认值快速开始开发

8、下面哪些项是常用SpringBoot配置文件类型？（A、B）

A. .properties B. .yml C. .config D. .ini

9、运行 Spring Boot 有哪几种方式（ABC）

A.打包用命令或者放到容器中运行

B.用 Maven/ Gradle 插件运行

C.直接执行 main 方法运行

10、关于启动SpirngBoot的方式正确的是：（D）

A:在idea中直接使用启动（最常用） B. 使用mvn 命令来启动

C. 使用mvn编译，而后在class目录生成jar包，使用Java命令来启动

D都对

1. 填空题（每空1分，共10分）
2. SpringBoot中外部配置支持使用\_\_\_\_\_\_\_\_\_\_文件，\_\_\_\_\_\_\_\_\_\_文件， \_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_， 使用\_\_\_\_\_\_\_\_\_\_注解， 可以直接将属性值注入到beans中。 (properties、YAML、环境变量、命令行参数、@Value)
3. Spring Boot 致力于快速产品就绪应用程序。为此，它提供了一些譬如\_\_\_\_\_\_，\_\_\_\_\_\_\_，监控和嵌入式服务器等开箱即用的非功能性特征。(高速缓存、日志记录)
4. springboot默认读取配置文件为\_\_\_\_\_\_\_\_\_\_\_或\_\_\_\_\_\_\_\_\_\_。(application.properties、application.yml)
5. Spring Boot已经建立在现有框架之上。使用 启动， （spring）
6. Spring Boot 可以不需要独立的容器运行因为内置了\_\_\_\_\_\_\_\_ 等容器。（Tomcat/ Jetty）
7. Spring Boot有一个开发工具（ ）模块，它有助于提高开发人员的生产力。Java开发人员面临的一个主要挑战是将文件更改自动部署到服务器并自动重启服务器。（DevTools）
8. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_该注解来表明该类是一个Spring的配置，相当于一个xml文件。(@Configuration)
9. SpringBoot核心配置文件中配置tomcat的默认端口的写法是server.\_\_\_\_\_\_\_。(port)
10. SpringBoot读取外部资源文件的注解是\_\_\_\_\_\_\_\_\_。（@PropertySource）
11. SpringBoot提供了\_\_\_\_\_\_\_\_\_\_\_\_操作来简化redis的使用。（RedisTempplate）
12. spring-boot-devtools插件的作用是开启\_\_\_\_\_\_\_\_\_。（热部署）
13. SpringCloud特点是：新建、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、组件支持丰富，功能齐全。(约定优于配置、开箱即用、快速启动)
14. 判断题
15. SpringBoot内置了Servlet容器，因此不能再将SpringBoot打包为WEB应用包在已经安装好的其他应用服务器中运行。（ F ）
16. Spring boot有效缓解了Spring 和 SpringMVC 需要配置大量参数的问题。（ T ）
17. Spring-boot-starter-web支持全栈式web开发，包括Tomcat和SpringMVC。（ T ）
18. Spring-boot-starter-data-jpa不支持Hibernate和MyBatis。( F： 支持spring-data-jpa、spring-orm，Hibernate )
19. Spring Security是通过configure完成用户授权的。（ T ）
20. Spring Security是通过大量的Filter实现的，Filter会在SpringMVC之后拦截请求并处理。（F : 在SpringMVC之前）
21. SpringMVC使用Commons FileUpLoad实现了一个MultipartResolver的实现类CommonsMultipartResolver，该实现类提供了文件上传支持。（ T ）
22. 创建SpringBoot项目有两种方法，一种是通过start.spring.io进行创建，另一种是通过创建Maven项目进行手动设置SpringBoot依赖。（ T ）
23. Spring Boot可以帮助我们以最多的工作量，更加健壮地使用现有的Spring功能。（ F ）
24. 开发人员可以重新加载Spring Boot上的更改，而无需重新启动服务器（ T ）
25. JavaConfig为开发人员提供了一种纯Java方法来配置与XML配置概念相似的Spring容器。（ T）
26. 对于集成Spring Boot和ActiveMQ，我们使用spring-boot-starter-activemq（T）
27. SpringBoot可以不借助外部容器独立运行。（ T ）
28. SpringBoot部署必须打包为war文件。（F）
29. SpringBoot内嵌了一个tomcat容器。（T）
30. SpringBoot建议继续使用传统jsp技术。（F）
31. SpringBoot也可以整合MyBatis框架。（T）
32. 默认情况下SpringBoot使用Logback作为日志框架？（ T ）

五、简答题

1、如何理解 Spring Boot 中的 Starters？

答案：Starters可以理解为启动器，它包含了一系列可以集成到应用里面的依赖包，你可以一站式集成 Spring 及其他技术，而不需要到处找示例代码和依赖包。如你想使用 Spring JPA 访问数据库，只要加入 spring-boot-starter-data-jpa 启动器依赖就能使用了。

Starters包含了许多项目中需要用到的依赖，它们能快速持续的运行，都是一系列得到支持的管理传递性依赖。

2、简要列举Spring Boot中的主要Starters

答案：

spring-boot-starter 核心Spring Boot starter， 包括自动配置支持， 日志和YAML

spring-boot-starter-actuator 生产准备的特性， 用于帮你监控和管理应用

spring-boot-starter-aop 对面向切面编程的支持， 包括 spring-aop 和AspectJ

spring-boot-starter-dataelasticsearch 对Elasticsearch搜索和分析引擎的支持， 包括spring-data-elasticsearch

spring-boot-starter-data-jpa 对"Java持久化API"的支持， 包括 spring-data-jpa ， spring-orm 和Hibernate

spring-boot-starter-data-rest 对通过REST暴露Spring Data仓库的支持， 通过 spring-data-rest-webmvc 实现

spring-boot-starter-hornetq 对"Java消息服务API"的支持， 通过HornetQ实现

spring-boot-starter-jtaatomikos 对JTA分布式事务的支持， 通过Atomikos实现

spring-boot-starter-redis 对REDIS键值数据存储的支持， 包括 spring-redis

spring-boot-starter-security 对 spring-security 的支持

spring-boot-starter-test 对常用测试依赖的支持， 包括JUnit, Hamcrest和Mockito， 还有 spring-test 模块

spring-boot-starter-web 对全栈web开发的支持， 包括Tomcat和 spring-webmvc

spring-boot-starter-websocket 对WebSocket开发的支持

spring-boot-starter-ws 对Spring Web服务的支持

3、JPA 和 Hibernate 有哪些区别？（5分）

答案：JPA 是一个规范或者接口

Hibernate 是 JPA 的一个实现

当我们使用 JPA 的时候，我们使用 javax.persistence 包中的注释和接口时，不需要使用 hibernate 的导入包。

我们建议使用 JPA 注释，因为哦我们没有将其绑定到 Hibernate 作为实现。后来（我知道 - 小于百分之一的几率），我们可以使用另一种 JPA 实现。

4、简述Spring Boot Starter的工作原理。（5分）

|  |
| --- |
| 答案：  1. Spring Boot 在启动时扫描项目所依赖的JAR包，寻找包含spring.factories文件的JAR  2. 根据spring.factories配置加载AutoConfigure类 。  3. 根据 @Conditional注解的条件，进行自动配置并将Bean注入Spring Context。 |

5、Spring Boot、Spring MVC 和 Spring 有什么区别？（5分）

|  |
| --- |
| 答案：   1. Spring Framework 最重要的特征是依赖注入。所有 SpringModules 不是依赖注入就是 IOC 控制反转。恰当使用 DI 或 IOC 可以开发松耦合应用。松耦合使得应用单元测试更加容易。 2. Spring MVC 提供了一种分离式的方法来开发 Web 应用。通过运用类似 DispatcherServelet，MoudlAndView 和 ViewResolver 等一些简单的概念，使得开发 Web 应用更加简单。 3. Spring 和 SpringMVC 的问题在于需要配置大量的参数。Spring Boot 通过一个自动配置和启动的项来目解决这个问题。为了更快的构建产品就绪应用程序，Spring Boot 提供了一些非功能性特征。同时SpringBoot更容易构建微服务项目 |

* 1. spring-boot-maven-plugin的作用是什么？（5分）

|  |
| --- |
| 答案：  spring-boot-maven-plugin提供类似 jar 打包或者运行应用程序的命令。  spring-boot：run 运行SpringBoot应用程序。  spring-boot：repackage 重新打包jar包或者war包使其可执行。  spring-boot：start 启动 SpringBoot 应用程序。  spring-boot：stop 停止SpringBoot 应用程序。  spring-boot：build-info 生成构建信息。 |

* 1. @RequestMapping 和 @GetMapping、@PostMapping 的区别？（5分）

|  |
| --- |
| 答案：  @RequestMapping注解将 HTTP 请求映射到 MVC 和 REST 控制器的处理方法上。  @GetMapping是一个组合注解，是@RequestMapping(method = RequestMethod.GET)的缩写。  @PostMapping是一个组合注解，是@RequestMapping(method = RequestMethod.POST)的缩写。 |

* 1. 列举常用SpringBoot依赖模块，并说明其主要功能？（5分）

|  |
| --- |
| 答案：   1. spring-boot-starter：核心启动器，包含自动配置、日志和YAML文件的支持。 2. spring-boot-starter-web：支持全栈web开发，包括tomcat和spring-webmvc。 3. spring-boot-starter-jdbc：支持JDBC数据库。 4. spring-boot-starter-test：支持常规的测试依赖。 5. spring-boot-starter-data-jpa：支持JPA。 6. spring-boot-starter-security：支持spring-security。 |

**9、**Spring Boot有哪些优点？

答：-快速创建独立运行的spring项目与主流框架集成

-使用嵌入式的servlet容器，应用无需打包成war包

-starters自动依赖与版本控制

-大量的自动配置，简化开发，也可修改默认值

-准生产环境的运行应用监控

-与云计算的天然集成

**10、如何使用Spring Boot实现分页和排序？**

答：

使用Spring Data-JPA可以实现将可分页的org.springframework.data.domain.Pageable传递给存储库方法。

11、运行 Spring Boot 有哪几种方式？（5分）

答案：

1）打包用命令或者放到容器中运行

2）用 Maven/ Gradle 插件运行

3）直接执行 main 方法运行

六、程序实现题

1、现有MySQL的建表语句如下：

CREATE TABLE `user` (

`id` int(3) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

使用Spring boot实现针对该表的crud操作

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.tianmaying</groupId>  <artifactId>spring-web-demo</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>spring-web-demo</name>  <description>Demo project for Spring WebMvc</description>  <!-- 父依赖 -->  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.2.5.RELEASE</version>  <relativePath/>  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <!-- mysql -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  </dependencies>  <build>    <plugins>    <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project>  ---------------------  @SpringBootApplication  @RestController  public class Application {    @RequestMapping("/")  public String greeting() {  return "Hello World!";  }  public static void main(String[] args) {  SpringApplication.run(Application.class, args);  }  ---------------------  @Entity  @Table(name = "user")  public class User {  @Id  @GeneratedValue(strategy = GenerationType.AUTO)  private long id;  @NotNull  private String name;    public User() {  }    public User(long id) {  this.id = id;  }  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Column(name = "ID", nullable = false, precision = 10, scale = 0)  public long getId() {  return id;  }    public void setId(long id) {  this.id = id;  }  @Column(name = "name")  public String getName() {  return name;  }    public void setName(String name) {  this.name = name;  }  ---------------------  @Transactional  public interface UserDao extends CrudRepository<User, Long> {  public User findByName(String name);  /\* public User save(User user);  public User update(User user);\*/  public void delete(User user);  ---------------------  @Controller  public class UserController {  @Autowired  public UserDao userDao;  @RequestMapping("/getbyname")  @ResponseBody  public String getByName(String name) {  String userId;  User user = userDao.findByName(name);  if (user != null) {  userId = String.valueOf(user.getId());  return "The user id is: " + userId;  }  return "user " + name + " is not exist.";  }    @RequestMapping("/save")  @ResponseBody  public String save() {  User user = new User();  user.setName("张三");  User user2 = userDao.save(user);  if (user != null) {  return "The user id is: " + user2.getId();  }  return "user id is not exist.";  }    @RequestMapping("/update")  @ResponseBody  public String update() {  User user = new User();  user.setId(1);  user.setName("李四");  User user2 = userDao.save(user);  if (user != null) {  return "The user id is: " + user2.getId();  }  return "user id is not exist.";  }    @RequestMapping("/delete")  @ResponseBody  public String delete() {  User user = new User();  user.setId(2);  userDao.delete(user);  return "执行了删除操作";  }  }  ---------------------  spring.datasource.url = jdbc:mysql://localhost:3306/springboot?useUnicode=true&characterEncoding=UTF-8  spring.datasource.username = root  spring.datasource.password =  spring.datasource.driverClassName = com.mysql.jdbc.Driver  # Specify the DBMS  spring.jpa.database = MYSQL  # Show or not log for each sql query  spring.jpa.show-sql = true  # Hibernate ddl auto (create, create-drop, update)  spring.jpa.hibernate.ddl-auto = update  # Naming strategy  spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy    # stripped before adding them to the entity manager)  spring.jpa.properties.hibernate.dialect =org.hibernate.dialect.MySQL5Dialect |

2、配置spring-boot连接到mysql数据库？（10分）

|  |
| --- |
| 第一步 - 把 mysql 连接器的依赖项添加至 pom.xml  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  第二步 - 从 pom.xml 中移除 H2 的依赖项  或者至少把它作为测试的范围。  <!--  <dependency>  <groupId>com.h2database</groupId>  <artifactId>h2</artifactId>  <scope>test</scope>  </dependency>  -->  第三步 - 安装你的 MySQL 数据库  更多的来看看这里 -https://github.com/in28minutes/jpa-with-hibernate#installing-and-setting-up-mysql  第四步 - 配置你的 MySQL 数据库连接  配置 application.properties  spring.jpa.hibernate.ddl-auto=none  spring.datasource.url=jdbc:mysql://localhost:3306/todo\_example  spring.datasource.username=todouser  spring.datasource.password=YOUR\_PASSWORD  第五步 - 重新启动，准备好了 |

3、实现SpringBoot2项目读写分离配置：假设数据库已经完成分库分表；使用Map集合接收从库数据；定义数据源时需要传入五个参数：数据源名称、主库数据源名称、主数据源、从数据源集合、访问策略。要是通过创建一个数据源配置类来实现（10分）

|  |
| --- |
| 答案（主要代码）：  @Configuration  public class DataSourceConfiguration {  @Autowired  private DBMasterConfig dbMasterConfig; //主数据库配置信息  @Autowired  private DBSlaveAConfig dbSlaveAConfig; //从数据库A配置信息  @Autowired  private DBSlaveBConfig dbSlaveBConfig; //从数据库B配置信息  @Bean  public DataSource getDataSource() throws SQLException {  return buildDataSource();  }  private DataSource buildDataSource() throws SQLException {  /\*\*设置从库数据源集合\*/  Map<String, DataSource> slaveDataSourceMap = new HashMap<>();  slaveDataSourceMap.put(  dbSlaveAConfig.getDatabaseName(),dbSlaveAConfig.createDataSource());  slaveDataSourceMap.put(dbSlaveBConfig.getDatabaseName(),  dbSlaveBConfig.createDataSource());  /\*\*获取数据源对象\*/  DataSource dataSource =  MasterSlaveDataSourceFactory.  createDataSource("masterSlave", dbMasterConfig.getDatabaseName()  , dbMasterConfig.createDataSource(), slaveDataSourceMap,  MasterSlaveLoadBalanceStrategyType.getDefaultStrategyType());  return dataSource;  }  @Bean  public KeyGenerator keyGenerator() {  return new DefaultKeyGenerator();  }  } |

4、实现注册和登录

（10分）

|  |
| --- |
| **配置pom.xml**  <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.loren</groupId>  <artifactId>LAR</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>LAR</name>  <description>Demo project for Spring Boot</description>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.8.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.16</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project>   * **编写application.yml，使项目连接到MySql数据库**   server:  port: 8080  context-path: /  spring:  datasource:  driver-class-name: com.mysql.jdbc.Driver  url: jdbc:mysql://localhost:3306/db\_user  username: root  password: 123456  jpa:  hibernate:  ddl-auto: update  show-sql: true   * **创建实体对象Use**   package com.loren.demo.entity;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.Id;  @Entity  public class User{  @Id  @GeneratedValue  private Long id;  private String username;  private String password;  private String tel;  private String realname;  public UserEntity() {}  public UserEntity(String username,String password) {  this.username=username;  this.password=password;  }  public Long getId() {  return id;  }  public void setId(Long id) {  this.id = id;  }  public String getUsername() {  return username;  }  public void setUsername(String username) {  this.username = username;  }  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  public String getTel() {  return tel;  }  public void setTel(String tel) {  this.tel = tel;  }  public String getRealname() {  return realname;  }  public void setRealname(String realname) {  this.realname = realname;  }  }   * **创建UserDao**   package com.loren.demo.dao;  import org.springframework.data.jpa.repository.JpaRepository;  import org.springframework.data.jpa.repository.Query;  import org.springframework.data.repository.CrudRepository;  import org.springframework.stereotype.Repository;  import com.loren.demo.entity.User;  @Repository  public interface UserDao extends JpaRepository<User, Long>{  @Query("select s from User s where s.username=?1 and s.password=?2")  public UserEntity findBynameAndPassword(String username,String password);  @Query("select s from User s where s.username=?")  public UserEntity findByName(String username);  }  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  创建UserController  package com.loren.demo.controller;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpSession;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import com.loren.demo.dao.UserDao;  import com.loren.demo.entity.UserEntity;  @Controller  @RequestMapping("/user/\*")  public class IndexController {  @Autowired  private UserDao userdao;  private User user=new User();  @RequestMapping("/index")  public String index(HttpSession httpSession) {  httpSession.getAttribute("userLogin");  return "index";  }  @RequestMapping("/register")  public String register() {  return "register";  }  @RequestMapping("/login")  public String login() {  String str="";  String username=user.getUsername();  if(username!=null) {  str="index";  }else {  str="login";  }  return str;  }  @RequestMapping("/uregister")  public String register(HttpServletRequest request) {  String username=request.getParameter("username");  String password=request.getParameter("password");  String password2=request.getParameter("password2");  String realname=request.getParameter("realname");  String str="";  if(password.equals(password2)) {  user=userdao.findByName(username);  if(user==null) {  User user=new User();  user.setUsername(username);  user.setPassword(password);  user.setRealname(realname);  userdao.save(user);  str="login";  }else {  str="register";  }  }else {  str="register";  }  return str;  }  @RequestMapping("/ulogin")  public String login(HttpServletRequest request,HttpSession session) {  String username=request.getParameter("username");  String password=request.getParameter("password");  user=userdao.findBynameAndPassword(username, password);  String str="";  if(user!=null) {  session.setAttribute("userLogin", user);  str="index";  }else {  str="login";  }  return str;  }  }   * **创建login.ftl**   <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8" />  <title>登陆</title>  </head>  <body>  <form action="/user/ulogin" method="post">  <input type="text" name="username" /></br>  <input type="password" name="password" /></br>  <input type="submit" value="登陆" /></br>  <a href="register.html" target="\_blank">注册</a>  </form>  </body>  </html>   * **创建register.ftl**   <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8" />  <title>注册</title>  </head>  <body>  <form action="/user/uregister" method="post">  用户名:<input type="text" name="username" /></br>  密码:<input type="password" name="password" /></br>  确认密码:<input type="password" name="password2" /></br>  真实姓名:<input type="text" name="realname" /></br>  <input type="submit" value="注册">  </form>  </body>  </html>   * **创建index.ftl**   <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8" />  <title>登陆成功</title>  </head>  <body>  欢迎您${userLogin.realname }  </body>  </html> |

5、用代码、命令演示SpringBoot配置数据库的配置?

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.example</groupId>  <artifactId>spring-boot-demo-28-1</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>spring-boot-demo-28-1</name>  <description>ehcache for Spring Boot</description>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.9.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-cache</artifactId>  </dependency>  <!-- cache -->  <dependency>  <groupId>net.sf.ehcache</groupId>  <artifactId>ehcache</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.1</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <!-- mail -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-mail</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

6、请编写以下SpringBoot配置文件内容： 连接本机mysql数据库，用户名root，密码123456，服务器端口8080，MyBatis的mapper映射文件在classpath路径下的mybatis目录中

（10分）

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost/myhr?useUnicode=true&characterEncoding=utf-8  spring.datasource.username=root  spring.datasource.password=123456  spring.datasource.driver-class-name=com.mysql.jdbc.Driver  server.port = 8080  mybatis.mapper-locations: classpath:mybatis/\*.xml |