**1、本章面试题**

       什么是springboot 它的主要作用

springboot如何整合mybatis

**2、知识点**

**2.1、课程回顾**

                shiro的web应用 和ssm框架的整合（认证和授权）

User user =(User) SecurityUtils.getSubject().getPrincpal(); //获取用户信息

SecurityUtils.getSubject().getSession().setAttribute("userInfo",user);

**2.2、本章重点**

         什么是springboot

springboot的特点

主要注解的解释

SpringApplication.run方法的执行

springboot整合mybatis 增删改查

**3、具体内容**

官网地址：<https://spring.io/projects/spring-boot/>

**3.1 什么是springboot**

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。

Spring Boot使您可以轻松地创建独立的、生产级的、基于Spring的应用程序，您可以“只是运行”。

我们对Spring平台和第三方库有一个独到的见解，这样您就可以从最少的麻烦开始了。大多数Spring引导应用程序只需要很少的Spring配置。

**3.2 特点**

1, 创建独立的Spring应用程序

2,直接嵌入Tomcat、Jetty或Undertow（不需要部署WAR文件）

3,提供独特的“starter”依赖项以简化构建配置

4,尽可能自动配置Spring和第三方库

5,提供生产就绪功能，如度量、运行状况检查和外部化配置

6,完全没有代码生成，也不需要XML配置

**3.3 Springboot  hello world实例**

**3.3.1 使用maven创建：**

   1) 创建maven项目

   2) pom中添加依赖jar

 <!-- Maven项目之间不仅存在多模块的聚合关系，而且Maven项目之间还可以存在相互继承的关  系。Maven项目之间的继承关系通过<parent>表示，当前项目则是spring-boot-starter-parent的一个子项目大部分配置都可以使用继承-->

    <parent>

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

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

        <version>2.0.4.RELEASE</version>

        <relativePath/>

    </parent>

    <dependencies>

            <!-- 核心模块，包括自动配置支持、日志支持 -->

        <dependency>

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

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

        </dependency>

    </dependencies>

3)创建HelloWorldController

4)创建启动类（springboot2.0以上,要求jdk1.8以上）

        @SpringBootApplication

public class SpringBootMainApplication {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

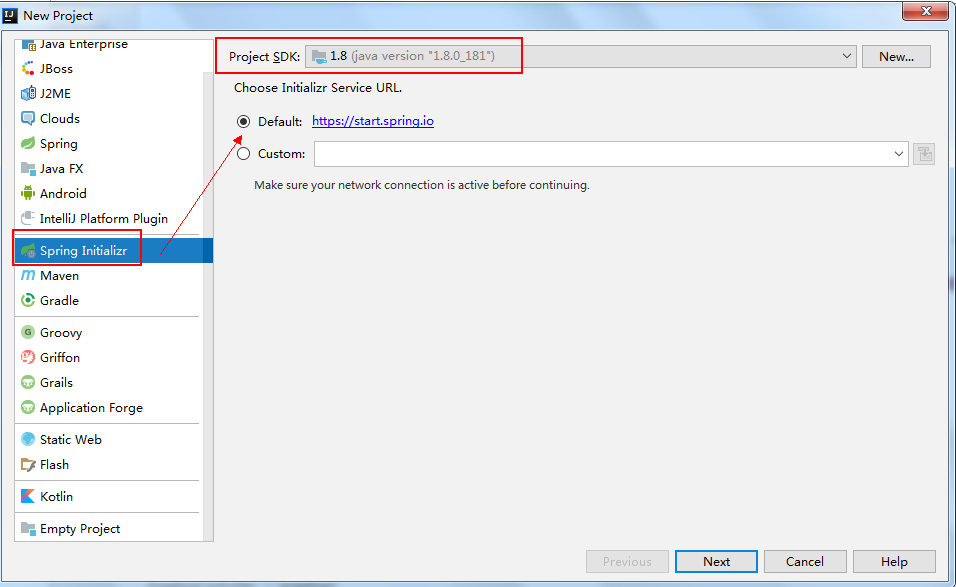
// TODO Auto-generated method stub

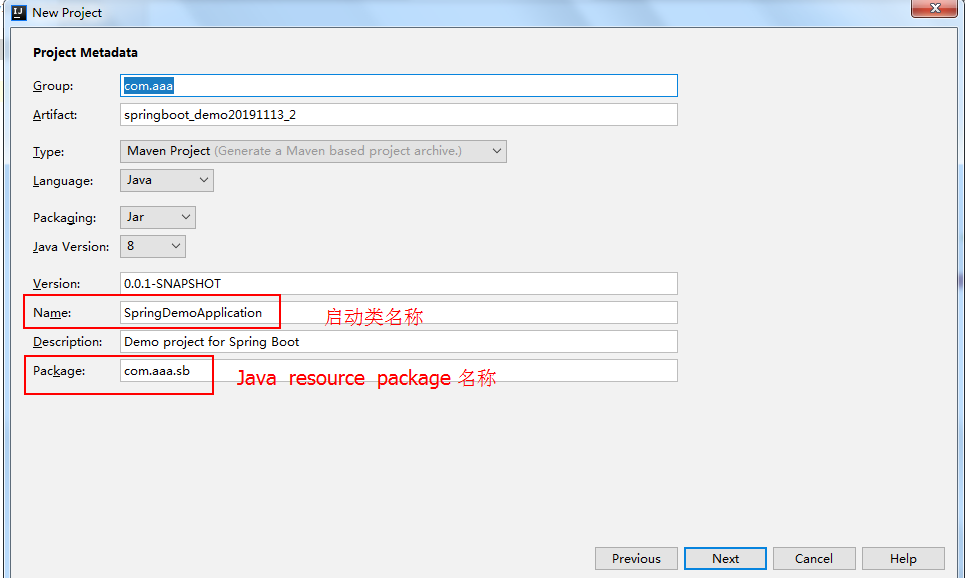
SpringApplication.run(SpringBootMainApplication.class, args);

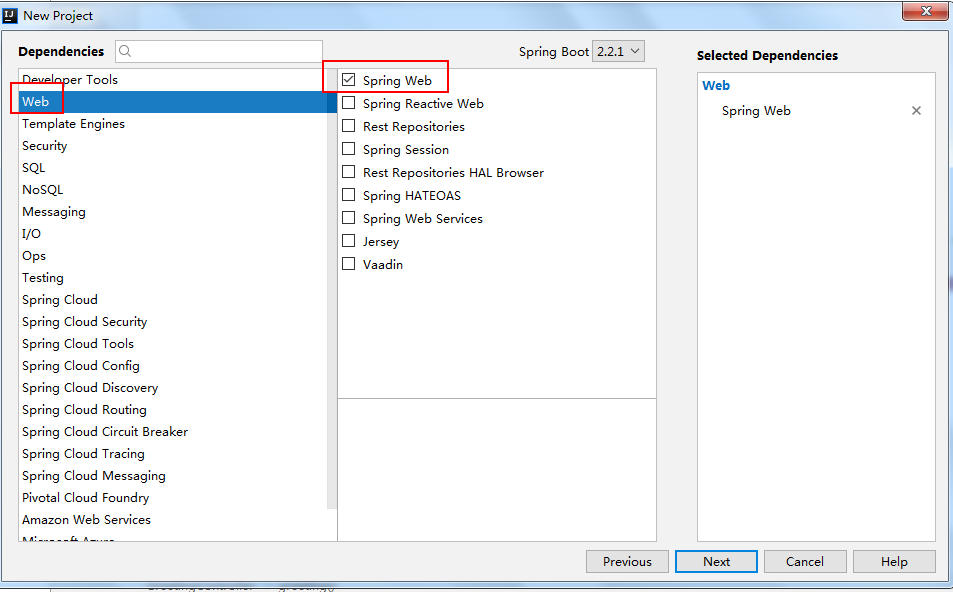
}

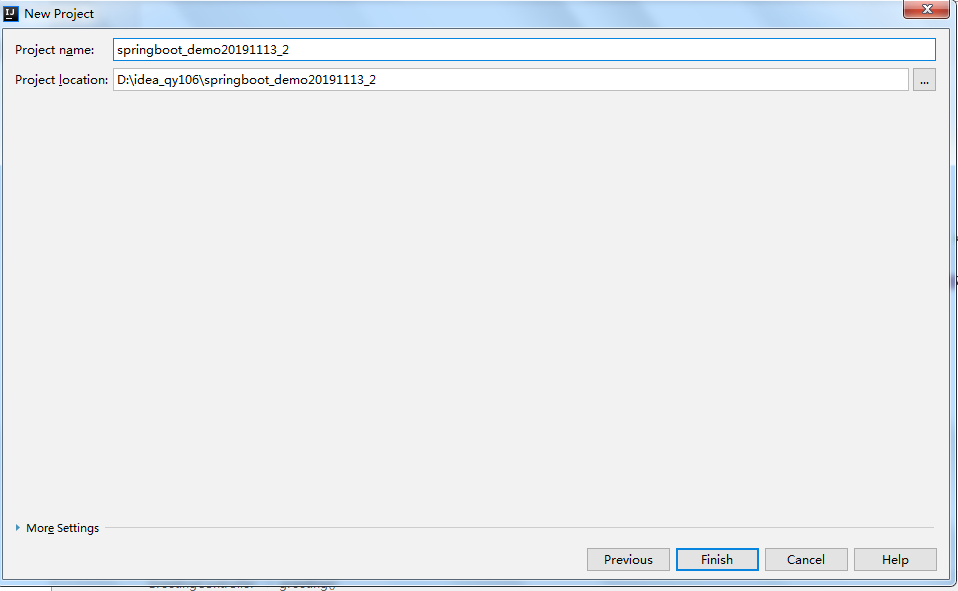
}

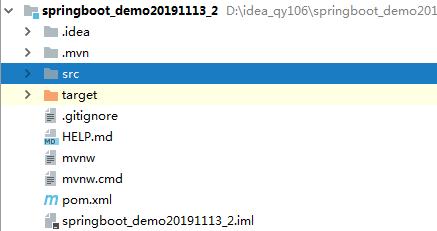
**3.3.2 使用idea工具简化创建**











**3.4，注解解释**

**@SpringBootApplication**

有了它项目就可以运行起来，因为他相当于3个注解的功能

@Configuration、@EnableAutoConfiguration、@ComponentScan

**@EnableAutoConfiguration** 。让Spring Boot根据类路径中的jar包依赖为当前项目进行自动配置。比如，添加了spring-boot-starter-web这个基础依赖，Spring Boot会自动添加tomcat和Spring MVC的依赖，并且自动对他们进行配置。

**@Configuration** 等同于spring的XML配置文件

**@ComponentScan** 组件扫描，可自动发现和装配一些Bean

**spring常用：**

**@ResponseBody：**表示该方法的返回结果直接写入HTTP response body中，一般在异步获取数据时使用，用于构建RESTful的api。在使用@RequestMapping后，返回值通常解析为跳转路径，加上@Responsebody后返回结果不会被解析为跳转路径，而是直接写入HTTP response body中。比如异步获取json数据，加上@Responsebody后，会直接返回json数据。

**@Controller：**用于定义控制器类，在spring项目中由控制器负责将用户发来的URL请求转发到对应的服务接口（service层），一般这个注解在类中，通常方法需要配合注解

**@Service：**一般用于修饰service层的组件

**@Repository：**使用@Repository注解可以确保DAO或者repositories提供异常转译，这个注解修饰的DAO或者repositories类会被ComponetScan发现并配置，同时也不需要为它们提供XML配置项。

**@RequestMapping** 提供路由信息，负责URL到Controller中的具体函数的映射。

**@RestController：**

       用于标注控制层组件(如struts中的action,springmvc的controller)，@ResponseBody和@Controller的合集。它继承自@Controller注解。spring4.0之前的版本，Spring MVC的组件都使用@Controller来标识当前类是一个控制器servlet。使用这个特性，我们可以开发REST服务的时候不需要使用@Controller而专门的@RestController。当你实现一个RESTful web services的时候，response将一直通过response body发送。为了简化开发，Spring 4.0提供了一个专门版本的controller。

**@Bean：**用@Bean标注方法等价于XML中配置的bean。

**@Value：**注入Spring boot application.properties配置的属性的值。

**@AutoWired：**自动导入依赖的bean。byType方式。把配置好的Bean拿来用，完成属性、方法的组装，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。当加上（required=false）时，就算找不到bean也不报错。

**@Qualifier：**当有多个同一类型的Bean时，可以用@Qualifier(“name”)来指定。与@Autowired配合使用。@Qualifier限定描述符除了能根据名字进行注入，但能进行更细粒度的控制如何选择候选者

**@Resource(name=”name”,type=”type”)**：没有括号内内容的话，默认byName@Autowired干类似的事。

 **3.5 SpringApplication.run:**

 创建SpringApplication对象；在对象初始化时保存事件监听器，容器初始化类以及判断是否为web应用，保存包含main方法的主配置类。

 调用run方法；准备spring的上下文，完成容器的初始化，创建，加载等。会在不同的时机触发监听器的不同事件。

  **源码：**

public ConfigurableApplicationContext run(String... args) {

    StopWatch stopWatch = new StopWatch();

    stopWatch.start();

    ConfigurableApplicationContext context = null;

    FailureAnalyzers analyzers = null;

    // 配置属性

    configureHeadlessProperty();

    // 获取监听器

    // 利用loadFactoryNames方法从路径MEAT-INF/spring.factories中找到所有的SpringApplicationRunListener

    SpringApplicationRunListeners listeners = getRunListeners(args);

    // 启动监听

    // 调用每个SpringApplicationRunListener的starting方法

    listeners.starting();

    try {

        // 将参数封装到ApplicationArguments对象中

        ApplicationArguments applicationArguments = new DefaultApplicationArguments(

            args);

        // 准备环境

        // 触发监听事件——调用每个SpringApplicationRunListener的environmentPrepared方法

        ConfigurableEnvironment environment = prepareEnvironment(listeners,

            applicationArguments);

        // 从环境中取出Banner并打印

        Banner printedBanner = printBanner(environment);

        // 依据是否为web环境创建web容器或者普通的IOC容器

        context = createApplicationContext();

        analyzers = new FailureAnalyzers(context);

        // 准备上下文

        // 1.将environment保存到容器中

        // 2.触发监听事件——调用每个SpringApplicationRunListeners的contextPrepared方法

        // 3.调用ConfigurableListableBeanFactory的registerSingleton方法向容器中注入applicationArguments与printedBanner

        // 4.触发监听事件——调用每个SpringApplicationRunListeners的contextLoaded方法

        prepareContext(context, environment, listeners, applicationArguments,

            printedBanner);

        // 刷新容器，完成组件的扫描，创建，加载等

        refreshContext(context);

        afterRefresh(context, applicationArguments);

        // 触发监听事件——调用每个SpringApplicationRunListener的finished方法

        listeners.finished(context, null);

        stopWatch.stop();

        if (this.logStartupInfo) {

            new StartupInfoLogger(this.mainApplicationClass)

                .logStarted(getApplicationLog(), stopWatch);

        }

        // 返回容器

        return context;

    }

    catch (Throwable ex) {

        handleRunFailure(context, listeners, analyzers, ex);

        throw new IllegalStateException(ex);

    }

}

**3.6，springboot整合mybatis**

**1）添加依赖**

             <dependency>

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

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

        </dependency>

      <!-- https://mvnrepository.com/artifact/org.mybatis.spring.boot/mybatis-spring-boot-starter -->

<dependency>

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

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

<version>2.1.1</version>

</dependency>

<!--oracle驱动包-->

<dependency>

<groupId>oracle</groupId>

<artifactId>oracle-jdbc</artifactId>

<version>12.1.0.2</version>

</dependency>

  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

    <dependency>

      <groupId>mysql</groupId>

      <artifactId>mysql-connector-java</artifactId>

      <version>5.1.10</version>

    </dependency>

 **2）添加  application.properties**

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

spring.datasource.username=scott

spring.datasource.password=tiger

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

#mybatis实体别名配置

mybatis.typeAliasesPackage=com.aaa.springboot.entity

#mapper文件扫描

mybatis.mapper-locations=classpath\:mapper/\*.xml

#工程名

server.servlet.context-path=/test

#端口号

server.port=8080

#tomcat编码

server.tomcat.uri-encoding=UTF-8

**3),编写dao及mapper**

**4),编写service及实现类**

**5),编写controller**

**6),编写页面**

**7),@MapperScan("com.aaa.sb.dao")**

**启动测试**

**RequestBody 不支持的问题，解决：**

**1，pom.xml添加包**

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.46</version>

</dependency>

2，创建配置类config.HttpConverterConfig 复制下面内容：

package com.aaa.sbm.config;

import com.alibaba.fastjson.serializer.SerializerFeature;

import com.alibaba.fastjson.support.config.FastJsonConfig;

import com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter;

import org.springframework.boot.autoconfigure.http.HttpMessageConverters;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.converter.HttpMessageConverter;

/\*\*

\* fileName:HttpConverterConfig

\* description:

\* author:zz

\* createTime:2019/11/25 17:24

\* version:1.0.0

\*/

@Configuration

public class HttpConverterConfig {

@Bean

public HttpMessageConverters fastJsonHttpMessageConverters() {

// 1.定义一个converters转换消息的对象

FastJsonHttpMessageConverter fastConverter = new FastJsonHttpMessageConverter();

// 2.添加fastjson的配置信息，比如: 是否需要格式化返回的json数据

FastJsonConfig fastJsonConfig = new FastJsonConfig();

fastJsonConfig.setSerializerFeatures(SerializerFeature.PrettyFormat);

// 3.在converter中添加配置信息

fastConverter.setFastJsonConfig(fastJsonConfig);

// 4.将converter赋值给HttpMessageConverter

HttpMessageConverter<?> converter = fastConverter;

// 5.返回HttpMessageConverters对象

return new HttpMessageConverters(converter);

}

}

3，测试成功

**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**