# ***全宇宙最精辟SpringBoot教程***

## **SpringBoot介绍**

## **创建第一个SpringBoot项目**

## **SpringBoot访问静态文件**

## **SpringBoot异常捕获**

## **SpringBoot集成thymleaf**

## **thymleaf展示数据**

## **SpringBoot集成JSP**

## **SpringBoot集成Mybatis**

## **SpringBoot事务管理**

## **SpringBoot多数据源**

## **SpringBoot集成Mybatis-Plus+代码生成**

## **SpringBoot集成lombok工具**

## **SpringBoot集成Shiro安全框架**

## **SpringBoot使用AOP统一处理请求日志**

## **SpringBoot定时任务@Scheduled**

## **SpringBoot异步调用Async**

## **SpringBoot自定义参数获取**

## **SpringBoot启动端口+访问路径**

## **SpringBoot配置文件yml**

## **SpringBoot打包发布**

## **SpringBoot多环境区分**

## **SpringBoot介绍**

第1次接触和学习Spring框架的时候，是不是需要配置很多的XML文件。当Spring越来越用上手的时候，有没有被这些繁琐的配置感到厌烦？每次新建项目都要进行反复的粘贴操作。如果你已经厌烦了这些操作，那你就必须来学习学习SpringBoot了。当然，如果你还喜欢那种配置，那注定你将与发展脱轨。

为什么我们要学习Springboot？原因是Spring Boot让我们的Spring应用变的更轻量化。比如：你可以仅仅依靠一个Java类来运行一个Spring引用。你也可以打包你的应用为jar并通过使用java -jar来运行你的Spring Web应用。

Spring Boot的主要优点：

为所有Spring开发者更快的入门

开箱即用，提供各种默认配置来简化项目配置

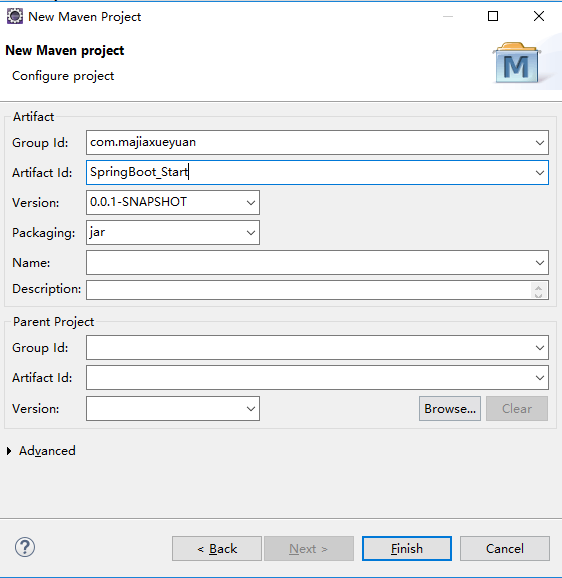
内嵌式容器简化Web项目

没有冗余代码生成和XML配置的要求

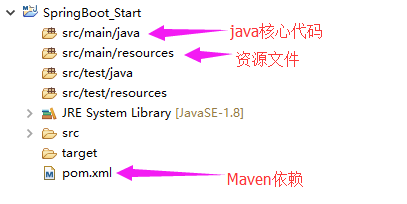
本章主要目标完成Spring Boot基础项目的构建，并且实现一个简单的Http请求处理，通过这个例子对Spring Boot有一个初步的了解，并体验其结构简单、开发快速的特性。

## **创建第一个SpringBoot项目**

新建一个Maven工程，没用过maven的同学可以自己下去看一下，我们现在只用到maven导包的功能，很简单



看一下项目结构：



**引入jar依赖：**

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.7.RELEASE</version>  <relativePath />  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

## **创建HelloController请求地址**

①：创建一个名为com.majiaxueyuan.controller的包

②：新建一个名为HelloController的Class

③：编写第一个请求hello

④：编写启动main方法

⑤：启动项目进行访问

**代码：**

看起来是不是很简单，完全抛弃了Spring最初的配置文件，也不需要Tomcat容器。完全做到了轻量化

|  |
| --- |
| @RestController  @EnableAutoConfiguration  public class HelloController {  @RequestMapping("/hello")  public String hello() {  return "Hello Majiaxueyuan!";  }  public static void main(String[] args) {  SpringApplication.run(HelloController.class, args);  }  } |

推荐启动方式：在com.majiaxueyuan下新建一个App类，在里面写main方法

**优化代码：**

HelloController.java

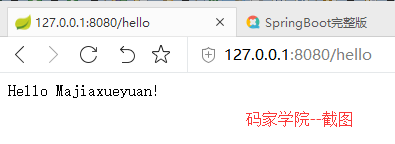
|  |
| --- |
| @RestController  public class HelloController {  @RequestMapping("/hello")  public String hello() {  return "Hello Majiaxueyuan!";  }  } |

App.java

|  |
| --- |
| @SpringBootApplication  public class App {  public static void main(String[] args) {  SpringApplication.run(App.class, args);  }  } |

这样的话方便以后我们扩展，实际开发中也是这样启动的，不推荐第一种启动方式

**浏览器访问效果**



## **SpringBoot访问静态文件**

所有的资源文件都应该在src/main/resources文件夹下面，但在SpringBoot中，系统默认扫描静态文件在static

或者public文件夹下，这里我们在src/main/resources目录下创建一个static文件夹，我们copy一张图片到static目录下，将图片命名为majiaxueyuan.png，我们启动项目并通过浏览器进行访问这张图片



在SpringBoot中访问静态资源是不是也很方便？完全不需要以前SSM框架或者SSH框架的繁琐。我知道这点完全不能满足你转投SpringBoot的欲望，那好，我们继续往下走。

## **SpringBoot异常捕获**

以前我们全局捕获异常得写切面去捕获，现在不用了，SpringBoot帮你封装好了，我们直接用就好了

@ExceptionHandler 表示拦截异常

@ControllerAdvice 是 controller 的一个辅助类，最常用的就是作为全局异常处理的切面类

@ControllerAdvice 可以指定扫描范围

@ControllerAdvice 约定了几种可行的返回值，如果是直接返回 model 类的话，使用 @ResponseBody 进行 json转换

我们在项目下面新建一个包com.majiaxueyuan.exception，新建一个异常拦截类GlobalExceptionHandler进行拦截Controller抛出的异常，我们这里拦截RuntimeException运行时异常，只需要在Controller里面制造一个异常，在hello请求里面增加 int a = 10 / 0;即可被捕获，其他的异常类型大家自己下去演示。

代码：

|  |
| --- |
| @ControllerAdvice  public class GlobalExceptionHandler {  @ExceptionHandler(RuntimeException.class)  @ResponseBody  public Map<String, Object> exceptionHandler() {  Map<String, Object> map = new HashMap<String, Object>();  map.put("errorCode", "404");  map.put("errorMsg", "找不到页面了!");  return map;  }  }  } |

## **SpringBoot集成thymleaf**

在之前的示例中，我们都是通过@RestController来处理请求，所以返回的内容为json对象。那么如果需要渲染html页面的时候，要如何实现呢？

我们说到了SpringBoot是SSM框架和SSH框架的替代品，那么毫无疑问也是用于搭建Web项目，前面简单了提到了控制层的使用和异常的捕获，这里，我们讲一下前台页面的展示。

SSM框架使用的JSP作为前台展示页面，大家是否知道JSP在运行时将要编译成一个Servlet，这样对服务器是一个大大的消耗。而今天，我们只需要使用前台模版引擎，这种方式就好比静态页面的方式，不需要占用服务器太多的资源，这又是我们转投SpringBoot的有一个不可抗拒的理由。

今天给大家介绍几种前台模版引擎：

Thymeleaf

FreeMarker

Groovy

这里介绍三种，这些都是SpringBoot官方推荐使用的模版，而且特别说出了避免使用JSP，目前用的最多的就是前两种，我这里就用Thymeleaf为大家讲我们的SpringBoot课程。若一定要使用JSP将无法实现Spring Boot的多种特性，后面我们会讲到集成JSP页面。

当我们使用上述模板引擎中的任何一个，SpringBoot默认的模板配置路径为：src/main/resources/templates。当然也可以修改这个路径，具体如何修改，可在后续各模板引擎的配置属性中查询并修改。所以今天我们需要在src/main/resources目录下创建一个templates文件夹用来存放我们的前台模版页面。

当使用模版的时候我们需要在配置文件中添加几条配置，SpringBoot自动会读取默认的配置文件路径是在resources下的application.properties文件，这里我们在配置文件中配置下面的参数

|  |
| --- |
| spring.thymeleaf.suffix=.html  spring.thymeleaf.mode=HTML5  spring.thymeleaf.encoding=UTF-8  spring.thymeleaf.content-type=text/html |

我们使用Thymeleaf的话需要引入相关的jar包

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId>  </dependency> |

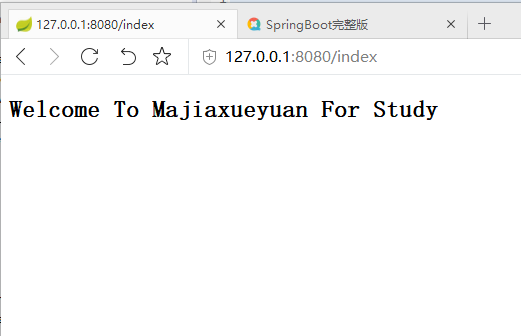
这里我们在前台页面目录templates下新建一个index.html文件，内容如下

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang="en">  <meta charset="UTF-8" />  <title></title>  </head>  <body>  <h2>Welcome To Majiaxueyuan For Study</h2>  </body>  </html> |

然后我们在com.majiaxueyuan.controller下新建一个Controller命名为CommonController，要返回到页面我们和SSM框架一样，在类上注解Controller而不是RestController，RestController代表是返回JSON字符串给前台，而Controller注解是返回页面，具体代码如下：

|  |
| --- |
| @Controller  public class CommonController {  @RequestMapping("/index")  public String index() {  return "index";  }  } |

然后这里就配置完成了我们的基本的页面和跳转，我们就可以通过浏览器进行访问了。

到这里我们访问127.0.0.1:8080/index就可以跳转到我们的index.html页面了。

## **thymleaf展示数据**

前面讲到了我们的视图跳转，下一步就是我们要将服务端的数据在页面上进行一个展示了，前台模版引擎一般有自己的规则，使用不同的模版引擎就有不同的规则需要去遵循才能正确的显示数据。

这里我随便列举几个展示例子，同学们自己用到什么模版引擎就自己下去简单搜索一下就好了，这个相对比较简单。

|  |
| --- |
| <input type="text" class="layui-input" th:value="${name}" />  <li th:text="${name}"></li>  <table border="1">   <tr>     <th>用户名</th>     <th>邮箱</th>   </tr>   <tr  th:each="user : ${userlist}">      <td th:text="${user.userName}">Onions</td>      <td th:text="${user.email}">test@test.com.cn</td>   </tr>  </table>  <li th:if="${name == null}">为空才显示</li>  <a th:></a> |

这四个是我们平时用的比较多的，具体更多的显示格式请自己下去查找，太多了，我就不一一演示了。

## **SpringBoot集成JSP**

如果有同学对JSP执迷不悟的话我们也是可以使用JSP的，当然如果使用JSP的话需要抛除我们上面引入thymeleaf的过程，我们这里就需要引入jsp的包

|  |
| --- |
| <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> |

然后在我们的配置文件中配置下面的参数，是不是感觉和SSM又有点相近了。

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

这样的话我们就可以像以前一样使用JSP进行访问了

## **SpringBoot集成Mybatis**

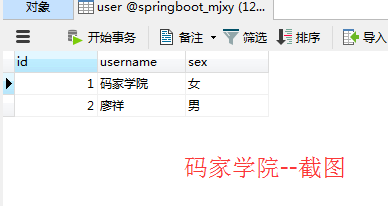
第一步我们需要在pom.xml里面引入mybatis相关的jar包

|  |
| --- |
| <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>  <version>5.1.21</version>  </dependency> |

在application.properties中填写我们的连接数据库相关的配置信息

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/springboot\_mjxy  spring.datasource.username=root  spring.datasource.password=admin  spring.datasource.driver-class-name=com.mysql.jdbc.Driver  #mybatis.mapper-locations=classpath:mapper/\*.xml |

这里首先我们在springboot\_mjxy数据库中新建一张user表，新增一条数据，结构如下：



然后我们新建一个com.majiaxueyuan.mapper的包，在包下新建一个Interface名为：UserMapper代码如下：注意使用Mapper注解让系统自己扫包到这里

|  |
| --- |
| @Mapper  public interface UserMapper {  @Select("select \* from user where id = #{id}")  User selectUserById(@Param("id") Integer id);  @Insert("INSERT INTO user (`id`, `username`, `sex`) VALUES (NULL, #{name}, #{sex});")  int insert(@Param("name") String name, @Param("sex") String sex);  } |

然后我们按照正常实际生产的方式来进行获取数据，新建Service层和实现层，这里就只贴实现层的代码如下：

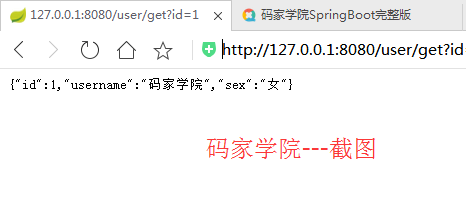
这里也需要Service注解，也是让系统自动扫包

|  |
| --- |
| @Service  public class UserServiceImpl implements UserService {  @Autowired  private UserMapper userMapper;  @Override  public User getUserById(Integer id) {  return userMapper.selectUserById(id);  }  @Override  public void insertUser(String name, String sex) {  userMapper.insert(name, sex);  }  } |

最后我们新建一个Controller进行访问数据，UserController代码如下：这里我们就用RestController注解返回JSON

|  |
| --- |
| @RestController  @RequestMapping("/user")  public class UserController {  @Autowired  private UserService userService;  @RequestMapping("/get")  public Object getUser(Integer id) {  return userService.getUserById(id);  }  @RequestMapping("/add")  public Object add(String name) {  System.out.println("add");  userService.insertUser(name, "SEX");  return "SUCCESS";  }  } |

## 最后将项目启动，什么都不用做变动，直接通过浏览器访问：http://127.0.0.1:8080/user/get?id=1，浏览器返回结果



到这里就集成完毕了我们的MyBatis，我们这里就不讲深入了，这里只要学习过SSM框架的都很简单了，后面我们会集成一个非常牛X的框架，Mybatis-Plus，到时候你们会发现完全不用自己写SQL了

## **SpringBoot事务管理**

什么是事物？？？

**是指多个SQL作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全地不执行。**

事物的四大特性（ACID）：

原子性：对于其数据修改，要么全都执行，要么全都不执行。

一致性：事务在完成时，必须使所有的数据都保持一致状态。

隔离性：由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。

持久性：事务完成之后，它对于系统的影响是永久性的。

比如说下面一系列操作（这里必须三条指令全部都执行完才能称为事物成功，只要有一条不成功，事物都不算成功，而且这里比如第三条信息失败了，第一条和第二条必须回滚）。

|  |
| --- |
| ①拿到一条数据  ②修改信息  ③更新回去 |

在SPringBoot中处理事务很简单，只需要一个注解，其他的SpringBoot自动帮我们实现。这里我们做一个简单的事物测试的例子，就是先将一个User的信息更新掉，然后再删除，我们在这两个过程中间制造一个异常看一下效果。

在UserMapper中新增两个SQL，一个更新，一个删除

|  |
| --- |
| @Update("UPDATE user SET username =#{name} WHERE id = #{id}")  int update(@Param("name") String name, @Param("id") Integer id);  @Delete("DELETE from user where id = #{id}")  void delete(@Param("id") Integer id); |

然后在Service中写一个事务测试方法

|  |
| --- |
| @Transactional  public void transcationtest(Integer id) {             // 我们要做的操作，我们将某个用户的NAME更新成待删除，然后去删除，但是中间我们制造一个异常出来  userMapper.update("DeleteTHIS", id);  System.out.println("异常之前");  int a = 2 / 0;  System.out.println("异常之后");  userMapper.delete(id);  } |

然后运行查看效果，发现Update根本不会成功，因为下面出现错误了，自动回滚，是不是很方便。这里顺便提一下数据缓存的操作，一般使用redis做数据缓存，这里画一下数据缓存的示例图。

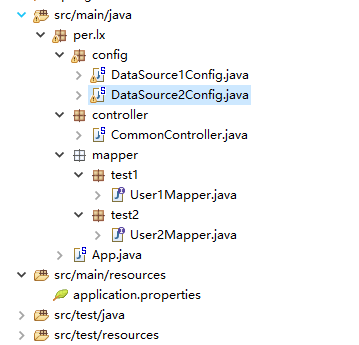
**SpringBoot多数据源**

我们这里集成一下Mybatis的多数据源情况。

什么是多数据源：多数据源指的是如果一个web项目用到了多个数据库，那么就需要连接多个数据库操作，这就是多数据源。

多数据源的好处：我要用到A数据库（mysql）中 的某些数据进行数据验证，并且要将数据插入B数据库中。这样做的好处是，不需要改变A中数据库的结构，就算是连接更多数据库的话，都不需要考虑其中结构问题，但是这就需要一个web项 目跟多个数据库进行连接操作的技术，而这个技术正是我今天介绍的多数据源配置。

**项目结构**



我们只需在配置文件中配置一下我们的两个数据源，然后分别写一个配置类加载进去就可以了。

看配置文件（代表两个数据源：用spring.datasource.test\*区别）：

|  |
| --- |
| spring.datasource.test1.driverClassName = com.mysql.jdbc.Driver  spring.datasource.test1.url = jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf-8  spring.datasource.test1.username = root  spring.datasource.test1.password = admin  spring.datasource.test2.driverClassName = com.mysql.jdbc.Driver  spring.datasource.test2.url = jdbc:mysql://localhost:3306/test2?useUnicode=true&characterEncoding=utf-8  spring.datasource.test2.username = root  spring.datasource.test2.password = admin |

然后为每个配置源配置一个加载类：

|  |
| --- |
| //这里配置mapper所在的包  @Configuration  @MapperScan(basePackages = "com.majiaxueyuan.mapper.test1", sqlSessionTemplateRef = "test1SqlSessionTemplate")  public class DataSource1Config {  @Bean(name = "test1DataSource")  @ConfigurationProperties(prefix = "spring.datasource.test1")            //primary代表主库  @Primary  public DataSource testDataSource() {  return DataSourceBuilder.create().build();  }  @Bean(name = "test1SqlSessionFactory")  @Primary  public SqlSessionFactory testSqlSessionFactory(@Qualifier("test1DataSource") DataSource dataSource)  throws Exception {  SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(dataSource);  return bean.getObject();  }  @Bean(name = "test1TransactionManager")  @Primary  public DataSourceTransactionManager testTransactionManager(@Qualifier("test1DataSource") DataSource dataSource) {  return new DataSourceTransactionManager(dataSource);  }  @Bean(name = "test1SqlSessionTemplate")  @Primary  public SqlSessionTemplate testSqlSessionTemplate(  @Qualifier("test1SqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception {  return new SqlSessionTemplate(sqlSessionFactory);  }  } |

这样的话就配置完成，可以调用不用的Mapper，实际上是存储在不同的库中。

这里大家会发现如果使用事务并且制造一个异常的话，会很明显的出现几种不同的效果，我们让mapper1和mapper2在不同顺序下面插入数据并且在中间抛出异常，发现只有一个插入成功了，另外一个没成功，为什么呢？就是因为@Primary的原因，这样就设置了主数据库。这里会发现设置了Primary的数据库只要出错都不会插入成功，都会事务回滚，大家注意了！！！！！

## **SpringBoot集成Mybatis-Plus+代码生成**

大家思考一个问题：平时业务代码不复杂的时候我们写什么代码写的最多，就是我们的SQL语句啊，配置那么多的Mapper.xml，还要配置什么resultMap这些东西，还要去管理paramtype。就会很容易出现错误，比如什么String can not parse to Integer之类的错误。

今天我们要学习一个超级牛X的框架，Mybatis-Plus，有了这个框架，我们可以做什么？不用写一句SQL，全部在JAVA代码中查找完毕，对于一些只做增删改查的管理系统，可以说业务层都可以免掉了。这样的框架是不是务必的强大，已经迫不及待的想要开始学习了，我们现在就正式开始。官网：http://baomidou.oschina.io/mybatis-plus-doc/、

首先MP是什么：是一个 [Mybatis](http://www.mybatis.org/mybatis-3/) 的增强工具，在 Mybatis 的基础上只做增强不做改变，为简化开发、提高效率而生。

MP特性：无侵入，依赖少，损耗小，防SQL注入，通用CRUD，多种主键策略，代码生成，内置分页插件。。。。

**通用CRUD**：集成BaseMapper就可以使用MP封装的CRUD

多种主键策略：IdType.AUTO（自动），IdType.INPUT（用户输入），IdType.ID\_WORKER（自动），IdType.UUID（自动）。配置方法，主键ID上加上注解：@TableId(value = "ID", type = IdType.AUTO)，一般情况下推荐大家使用自动增长主键。

**内置分页插件**：Page内置分页插件。

**代码生成**：MP自带代码生成工具，可以从Controller层直接生成到mapper层，包括实体类，让我们只关心请求地址和业务处理。

集成Mybaits-Plus

在pom.xml中导入相关jar包

|  |
| --- |
| <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus</artifactId>  <version>2.1.4</version>  </dependency> |

然后新建一个com.majiaxueyuan.config用来存储配置信息类的包，新建一个配置Mybatis-Plus的类MybatisPlusConfig

具体内容如下，同学们不需要记住配置的具体内容是什么，你大概能看懂是什么意思就懂了。

在SpringBoot中使用配置文件时，务必在类上加上Configuration的注解方便系统启动时加载。

|  |
| --- |
| @Configuration  public class MybatisPlusConfig {  @Autowired     private DataSource dataSource;     @Autowired     private MybatisProperties properties;     @Autowired     private ResourceLoader resourceLoader = new DefaultResourceLoader();     @Autowired(required = false)     private Interceptor[] interceptors;     @Autowired(required = false)     private DatabaseIdProvider databaseIdProvider;     /\*\*      \*   mybatis-plus分页插件      \*/     @Bean     public PaginationInterceptor paginationInterceptor() {         PaginationInterceptor page = new PaginationInterceptor();         page.setDialectType("mysql");         return page;     }     /\*\*      \* 这里全部使用mybatis-autoconfigure 已经自动加载的资源。不手动指定      \* 配置文件和mybatis-boot的配置文件同步      \* @return      \*/     @Bean     public MybatisSqlSessionFactoryBean mybatisSqlSessionFactoryBean() {         MybatisSqlSessionFactoryBean mybatisPlus = new MybatisSqlSessionFactoryBean();         mybatisPlus.setDataSource(dataSource);         mybatisPlus.setVfs(SpringBootVFS.class);         if (StringUtils.hasText(this.properties.getConfigLocation())) {             mybatisPlus.setConfigLocation(this.resourceLoader.getResource(this.properties.getConfigLocation()));         }         mybatisPlus.setConfiguration(properties.getConfiguration());         if (!ObjectUtils.isEmpty(this.interceptors)) {             mybatisPlus.setPlugins(this.interceptors);         }         MybatisConfiguration mc = new MybatisConfiguration();         mc.setDefaultScriptingLanguage(MybatisXMLLanguageDriver.class);         mybatisPlus.setConfiguration(mc);         if (this.databaseIdProvider != null) {             mybatisPlus.setDatabaseIdProvider(this.databaseIdProvider);         }         if (StringUtils.hasLength(this.properties.getTypeAliasesPackage())) {             mybatisPlus.setTypeAliasesPackage(this.properties.getTypeAliasesPackage());         }         if (StringUtils.hasLength(this.properties.getTypeHandlersPackage())) {             mybatisPlus.setTypeHandlersPackage(this.properties.getTypeHandlersPackage());         }         if (!ObjectUtils.isEmpty(this.properties.resolveMapperLocations())) {             mybatisPlus.setMapperLocations(this.properties.resolveMapperLocations());         }         return mybatisPlus;     }  } |

到这里，Mybatis-Plus已经全部集成到系统里面去了，然后我们就来改造我们之前的UserMapper，这里也可以不是说改造，就是删除掉里面的内容然后继承至Mybatis-Plus相关内容就可以了，代码如下：

|  |
| --- |
| @Mapper  public interface UserMapper extends BaseMapper<User> {  } |

你没有看错，同学们，我们之前的四个方法已经被我们改造来一个都没有，你要相信自己的眼睛，然后我们改造一下我们的Service层，只需要将里面调用的方法进行一个更改就可以了。

然后我们需要对实体类进行一个简单的修饰，并且告诉MP该表的主键是什么，然后主键的生成策略是什么。

|  |
| --- |
| @TableName("user")  public class User {  @TableId(value = "id", type = IdType.AUTO)  private Integer id;  private String username;             。。。。。。 |

然后我们这里来执行一下条件查询，这里我们要用到Mybatis-Plus中的包装（Wrapper）去构建我们的条件查询

下面举几个Wrapper语句的示例，

|  |
| --- |
| Wrapper<T> wrapper = new EntityWrapper<>();           构建一个实体类的包装工具  wrapper.eq("username", "LIAOXIANG");                          做条件判断  wrapper.between("id", 0, 100);                                         做范围判断  wrapper.groupBy("username");                                        分组  wrapper.isNotNull("username");                                        不为空判断  wrapper.orderBy("id", false);                                             排序，从小打大为true，反之false  。。。。。。。。。。  还有很多API，我这里就不一一给大家做讲解了，我们今天就用几个来演示一下，大家下去自己研究研究更多的功能 |

**代码生成工具**

使用我们的代码生成工具，需要导入一个模版引擎：

|  |
| --- |
| <dependency>  <groupId>org.apache.velocity</groupId>  <artifactId>velocity-engine-core</artifactId>  <version>2.0</version>  </dependency> |

代码我直接发给大家，然后大家下去跟着我自己练习一下就可以了。

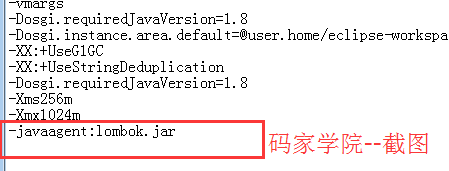
## **SpringBoot集成lombok工具(生成setget方法，构造函数，**打印日志**)**

平时要去打印日志

Logger log = LoggerFactory.getLogger(Class.class);

log.info();

平时我们写的一些重复代码，比如每个实体类的setter，getter方法，给每个类写上Logger获取的方法，这样写的话太繁琐，我们就可以使用lombok的工具去简化这个配置的操作。首先先要在Eclipse下安装lombok，在控制台中，使用java -jar lombok.jar为eclipse安装lombok插件，安装成功后需要重启eclipse，在eclipse目录下可以看到lombok.jar这个jar包，然后我们打开eclipse.ini配置文件查看到如下代码就正确



然后我们需要在pom.xml中引入lombok的jar包了

|  |
| --- |
| <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </dependency> |

到这里lombok就集成完毕了，我说过，我们要做最牛X的SpringBoot的教程，下面我们就看看这个工具的牛X之处

这里是一些常用的注解，还有更多的请大家参见官网：[http://projectlombok.org/features/index.](http://projectlombok.org/features/index.html)

|  |
| --- |
| @Data   ：注解在类上；提供类所有属性的 getter 和 setter 方法，此外还提供了equals、canEqual、hashCode、toString 方法  @Setter：注解在属性上；为属性提供 setting 方法  @Getter：注解在属性上；为属性提供 getting 方法  @Log4j ：注解在类上；为类提供一个 属性名为log 的 log4j 日志对象  @NoArgsConstructor：注解在类上；为类提供一个无参的构造方法  @AllArgsConstructor：注解在类上；为类提供一个全参的构造方法 |

## **SpringBoot集成Shiro安全框架**

Apache Shiro是Java的一个安全框架。目前，使用Apache Shiro的人越来越多，因为它相当简单，对比Spring Security，可能没有Spring Security做的功能强大，但是在实际工作时可能并不需要那么复杂的东西，所以使用小而简单的Shiro就足够了。对于它俩到底哪个好，这个不必纠结，能更简单的解决项目问题就好了。

Shiro有哪些功能：身份认证/登录，授权，会话管理，加密，**Web支持。。。。**

**记住这么一点，Shiro不会去维护用户、维护权限；这些需要我们自己去设计/提供；通过Realm让开发人员自己注入。**

**首先引入shiro相关jar包**

|  |
| --- |
| <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-spring</artifactId>  <version>1.4.0</version>  </dependency>  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-ehcache</artifactId>  <version>1.4.0</version>  </dependency> |

## 然后在src/main/resources下新建一个关于shiro的缓存配置文件ehcache-shiro.xml

|  |
| --- |
| **<?xml version="1.0" encoding="UTF-8"?>**  <ehcache updateCheck="false" name="cacheManagerConfigFile">  <defaultCache            maxElementsInMemory="10000"            eternal="false"             timeToIdleSeconds="120"            timeToLiveSeconds="120"            overflowToDisk="false"            diskPersistent="false"            diskExpiryThreadIntervalSeconds="120"            memoryStoreEvictionPolicy="LRU"/>   <cache name="shiro-activeSessionCache"     eternal="false"          maxElementsInMemory="10000"          overflowToDisk="false"           timeToIdleSeconds="0"          timeToLiveSeconds="0"          statistics="true"/>  </ehcache> |

然后新建一个Shiro的配置文件在com.majiaxueyuan.config包下ShiroConfig并注解@Configuration，这个配置只不过把最初的spring-shiro.xml形式的配置文件转成了java文件，这个大家自己拿下去看下就明白意思了，内容和xml形式完全一致

|  |
| --- |
| @Configuration  public class ShiroConfiguration {  /\*\*  \* ShiroFilterFactoryBean 处理拦截资源文件问题。  \* 注意：单独一个ShiroFilterFactoryBean配置是或报错的，以为在  \* 初始化ShiroFilterFactoryBean的时候需要注入：SecurityManager  \* Filter Chain定义说明 1、一个URL可以配置多个Filter，使用逗号分隔 2、当设置多个过滤器时，全部验证通过，才视为通过  \* 3、部分过滤器可指定参数，如perms，roles  \*/  @Bean  public ShiroFilterFactoryBean shirFilter(org.apache.shiro.mgt.SecurityManager securityManager) {  ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();  // 必须设置 SecurityManager  shiroFilterFactoryBean.setSecurityManager(securityManager);  // 拦截器.  Map<String, String> filterChainDefinitionMap = new LinkedHashMap<String, String>();  //配置静态资源允许访问  filterChainDefinitionMap.put("/js/\*\*","anon");  filterChainDefinitionMap.put("/css/\*\*","anon");  filterChainDefinitionMap.put("/index","anon");  // <!-- authc:所有url都必须认证通过才可以访问; anon:所有url都都可以匿名访问-->  filterChainDefinitionMap.put("/\*\*", "authc");  // 如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面  shiroFilterFactoryBean.setLoginUrl("/login");  // 未授权界面;  shiroFilterFactoryBean.setUnauthorizedUrl("/403");  Map<String, Filter> filters=new HashMap<String,Filter>();  shiroFilterFactoryBean.setFilters(filters);  shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);  return shiroFilterFactoryBean;  }  @Bean  public EhCacheManager getEhCacheManager() {  EhCacheManager em = new EhCacheManager();  em.setCacheManagerConfigFile("classpath:ehcache-shiro.xml");  return em;  }             // 开启Controller中的shiro注解  @Bean  public DefaultAdvisorAutoProxyCreator getDefaultAdvisorAutoProxyCreator() {  DefaultAdvisorAutoProxyCreator daap = new DefaultAdvisorAutoProxyCreator();  daap.setProxyTargetClass(true);  return daap;  }  /\*\*  \* 配置org.apache.shiro.web.session.mgt.DefaultWebSessionManager  \* @return  \*/  @Bean  public DefaultWebSessionManager getDefaultWebSessionManager(){  DefaultWebSessionManager defaultWebSessionManager=new DefaultWebSessionManager();  defaultWebSessionManager.setSessionDAO(getMemorySessionDAO());  defaultWebSessionManager.setGlobalSessionTimeout(4200000);  defaultWebSessionManager.setSessionValidationSchedulerEnabled(true);  defaultWebSessionManager.setSessionIdCookieEnabled(true);  defaultWebSessionManager.setSessionIdCookie(getSimpleCookie());  return defaultWebSessionManager;  }  /\*\*  \* 配置org.apache.shiro.session.mgt.eis.MemorySessionDAO  \* @return  \*/  @Bean  public MemorySessionDAO getMemorySessionDAO(){  MemorySessionDAO memorySessionDAO=new MemorySessionDAO();  memorySessionDAO.setSessionIdGenerator(javaUuidSessionIdGenerator());  return memorySessionDAO;  }  @Bean  public JavaUuidSessionIdGenerator javaUuidSessionIdGenerator(){  return new JavaUuidSessionIdGenerator();  }  /\*\*  \* session自定义cookie名  \* @return  \*/  @Bean  public SimpleCookie getSimpleCookie(){  SimpleCookie simpleCookie=new SimpleCookie();  simpleCookie.setName("security.session.id");  simpleCookie.setPath("/");  return simpleCookie;  }  @Bean  public LifecycleBeanPostProcessor getLifecycleBeanPostProcessor(){  return new LifecycleBeanPostProcessor();  }             @Bean(name = "securityManager")             public DefaultWebSecurityManager getDefaultWebSecurityManager(UserRealm userRealm) {                         DefaultWebSecurityManager dwsm = new DefaultWebSecurityManager();                         dwsm.setRealm(userRealm);                          //  <!-- 用户授权/认证信息Cache, 采用EhCache 缓存 -->                         dwsm.setCacheManager(getEhCacheManager());                         dwsm.setSessionManager(getDefaultWebSessionManager());                         return dwsm;               }  @Bean  public UserRealm userRealm(EhCacheManager cacheManager) {  UserRealm userRealm = new UserRealm();  userRealm.setCacheManager(cacheManager);  return userRealm;  }  /\*\*  \* 开启shrio注解支持  \* @param userRealm  \* @return  \*/  @Bean  public AuthorizationAttributeSourceAdvisor getAuthorizationAttributeSourceAdvisor(UserRealm userRealm){  AuthorizationAttributeSourceAdvisor aasa=new AuthorizationAttributeSourceAdvisor();  aasa.setSecurityManager(getDefaultWebSecurityManager(userRealm));  return aasa;  }  } |

## 然后在com.majiaxueyuan.realm写一个UserRealm做登录权限控制：

|  |
| --- |
| public class UserRealm extends AuthorizingRealm {  @Override  protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  // 这里做权限控制  return null;  }             @Override  protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {  // 这里做登录控制  return null;  }  } |

我们这里就在UserContrller里面去使用我们的shiro进行登录权限控制

|  |
| --- |
| UsernamePasswordToken token = new UsernamePasswordToken(username, password);  Security sccurity = SecurityUtils.getSubject();  sccurity.login(token); |

这里如果不抛出异常证明登录成功。抛出异常则证明账号或者密码错误。

这里，基本上shiro的集成使用就到这里了，关于shiro的基础不是我们要用到了，我们这里只是着重的要集成shiro到我们的SpringBooot里面去。完全不会shiro的请自行查看资料学习基础，请谅解。

## **SpringBoot使用AOP统一处理请求日志**

这里就提到了我们Spring当中的AOP，也就是面向切面编程，今天我们使用AOP去对我们的所有请求进行一个统一处理。首先在pom.xml中引入我们需要的aop的jar包

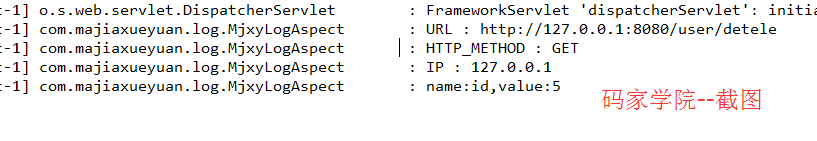
|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-aop</artifactId>  </dependency> |

然后我们新建一个Aspect去对我们的所有Controller进行一个日志拦截。在SPringBoot中我们使用AOP也是很简单的，只需要在类上加上一个@Aspect的注解就好了，然后通过@Component注解到我们的Spring容器中去。

我们新建一个包com.majiaxueyuan.log专门用来放我们的日志记录，代码如下：

|  |
| --- |
| @Aspect  @Component  @Slf4j  public class MjxyLogAspect {  @Pointcut("execution(public \* com.majiaxueyuan.controller..\*.\*(..))")  public void webLog() {  }  @Before("webLog()")  public void doBefore(JoinPoint joinPoint) throws Throwable {  // 接收到请求，记录请求内容  ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();  HttpServletRequest request = attributes.getRequest();  // 记录下请求内容  log.info("URL : " + request.getRequestURL().toString());  log.info("HTTP\_METHOD : " + request.getMethod());  log.info("IP : " + request.getRemoteAddr());  Enumeration<String> enu = request.getParameterNames();  while (enu.hasMoreElements()) {  String name = (String) enu.nextElement();  log.info("name:{},value:{}", name, request.getParameter(name));  }  }  @AfterReturning(returning = "ret", pointcut = "webLog()")  public void doAfterReturning(Object ret) throws Throwable {  // 处理完请求，返回内容  log.info("RESPONSE : " + ret);  }  } |

然后我们通过浏览器随便请求一次，可以看到控制台输出，这里我们就可以在实际生产环境中存储日志。



我们码家学院只做牛X的东西。或者说与牛X挂钩的

## **SpringBoot定时任务@Scheduled**

这个到了后面就比较简单了，我们就简单的写一个例子给大家展示。

在service中新建一个类ScheduledTest，代码如下：

|  |
| --- |
| @Component  public class ScheduledTest {  private static final SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");  @Scheduled(fixedRate = 2000)  public void scheduled() {  System.out.println("码家学院提示你==》现在时间：" + dateFormat.format(new Date()));  }  } |

另外我们需要在主函数启动类上开启定时器

|  |
| --- |
| @SpringBootApplication  @EnableScheduling  public class App {  public static void main(String[] args) {  SpringApplication.run(App.class, args);  }  } |

## **SpringBoot异步调用Async**

这个和定时器差不多，启动加上@EnableAsync ，只需要在我们需要异步的方法上面加上@Async注解

## **SpringBoot自定义参数获取**

在配置文件中我们可以去在配置文件中自定义一些参数，比如：

|  |
| --- |
| name=majiaxueyuan |

我们在代码中获取到我们的这个配置文件

|  |
| --- |
| @Value("${name}")  private String name;  @ResponseBody  @RequestMapping("/getValue")  public String getValue() {  return name;  } |

## **SpringBoot启动端口+访问路径**

## **只需要在配置文件中添加访问端口和访问路径就OK，一般添加在配置文件开头**

|  |
| --- |
| server.port=9090  server.context-path=/springboot |

现在只能用http://127.0.0.1:9090/springboot才能访问到项目

## **SpringBoot配置文件yml**

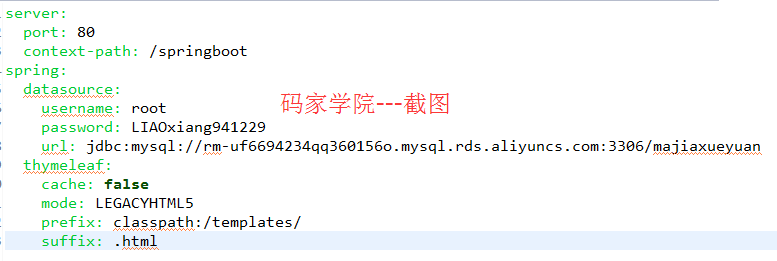
## 下面讲一下SpringBoot中另外一种格式的配置文件，名为application.yml的配置文件，这种配置文件更方便我们使用，有提示功能，而且SpringBoot也是默认去读取这个格式的配置文件，我们这里改变一下配置文件的风格。

下面是我们整套课程完整的配置文件：

|  |
| --- |
| server.port=9090  server.context-path=/springboot  spring.datasource.url=jdbc:mysql://localhost:3306/springboot\_mjxy  spring.datasource.username=root  spring.datasource.password=admin  spring.datasource.driver-class-name=com.mysql.jdbc.Driver  #mybatis.mapper-locations=classpath:mapper/\*.xml  spring.thymeleaf.suffix=.html  spring.thymeleaf.mode=HTML5  spring.thymeleaf.encoding=UTF-8  spring.thymeleaf.content-type=text/html  name=majiaxueyuan |

下面我们把它改造成application.yml的风格，非常有层次感

|  |
| --- |
| server:   port: 80   context-path: /springboot  spring:   datasource:     username: root     password: admin     url: jdbc:mysql://localhost:3306/springboot\_mjxy?characterEncoding=utf8   thymeleaf:     cache: false     mode: LEGACYHTML5     prefix: classpath:/templates/     suffix: .html |



如果没有高亮，请在eclipse中安装SpringToolsSuits。然后重启eclipse，选择打开方式就可以高亮了。

**SpringBoot多环境区分**

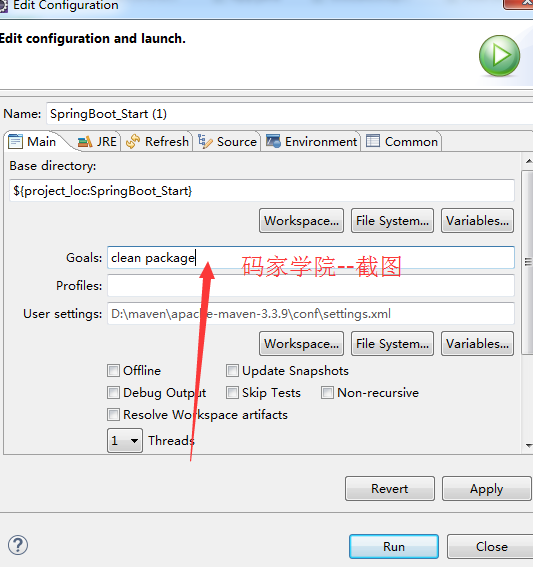
**我们可以创建多个配置文件，只需要在源配置文件上面增加下面的一个配置**

|  |
| --- |
| spring:   profiles:     active: lx |

这样系统就会优先去扫描配置文件是application-lx.yml的配置文件

## **SpringBoot打包发布**

项目右键-->Run As..-->Maven Build...



如果报错没有主清单,在pom文件中新增下面代码

|  |
| --- |
| **<build>**  **<plugins>**  **<plugin>**  **<groupId>org.apache.maven.plugins</groupId>**  **<artifactId>maven-compiler-plugin</artifactId>**  **<configuration>**  **<source>1.8</source>**  **<target>1.8</target>**  **</configuration>**  **</plugin>**  **<plugin>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-maven-plugin</artifactId>**  **<configuration>**  **<maimClass>com.majiaxueyuan.App</maimClass>**  **</configuration>**  **<executions>**  **<execution>**  **<goals>**  **<goal>repackage</goal>**  **</goals>**  **</execution>**  **</executions>**  **</plugin>**  **</plugins>**  **</build>** |

到这里，我们全宇宙最精辟的SpringBoot教程就结束了，我相信你已经在这里学习到了怎么去使用，怎么快捷的使用，怎么采用当下最牛X的框架去使用SpringBoot，后续我们将退出更多的中高级课程。包含多线程，高并发，高可用，分布式，微服务。感谢你们选择码家学院。

**QQ：438944209**

**网址：www.majiaxueyuan.com**