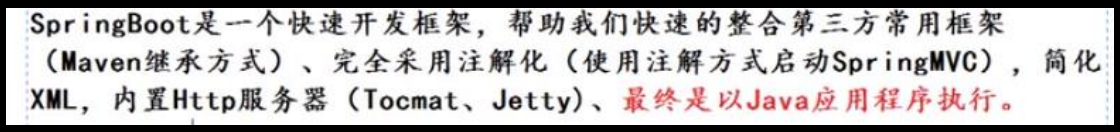
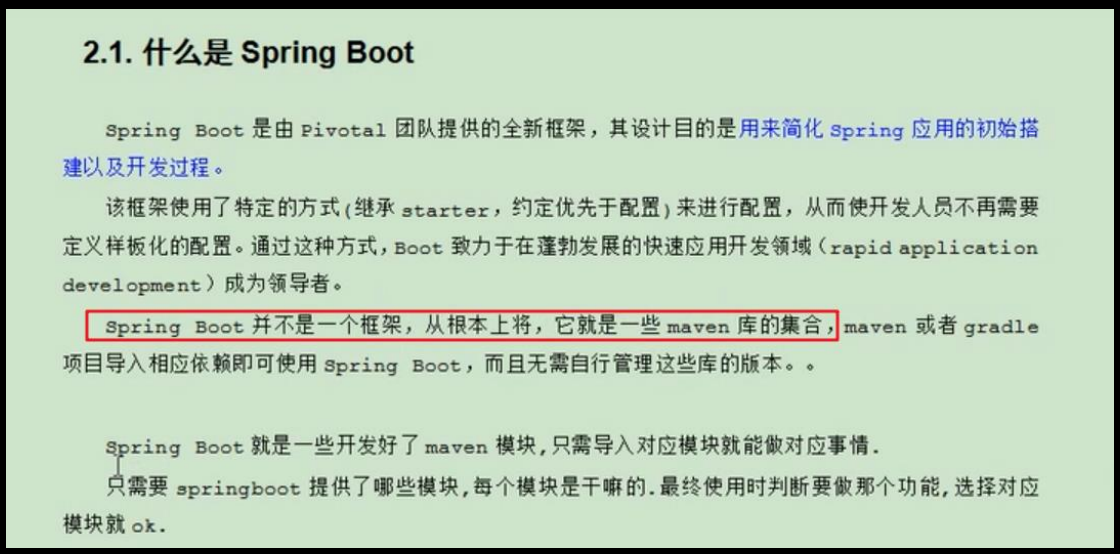
SpringBoot 学习笔记

- spring mvc < spring <springboot
- 微服务：把大的项目分成多个小型项目（微服务）
- Boot 作用：
    1. 简化 j2ee 开发
    2. 整个 Spring 技术栈的整合（mybiti mvc redis）
    3. 整个 J2EE 的整合

传统项目存在的问题：配置文件、jar 冲突

- Springboot 是快速开发框架，能快速整合第三方框架（Maven、Tomcat、jetty），完全使用注解化，没有 web.xml。
- Springboot 默认使用 Tomcat 服务器。
- 采用注解方式启动 mvc，内置嵌入 HTTP 服务器。
- 最终以 Java application 执行
- 采用 Maven 继承方式整合。





2.1. 什么是 Spring Boot

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

该框架使用了特定的方式(继承 starter，约定优先于配置)来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Boot 致力于在蓬勃发展的快速应用开发领域（rapid application development）成为领导者。

Spring Boot 并不是一个框架，从根本上将，它就是一些 maven 库的集合，maven 或者 gradle 项目导入相应依赖即可使用 Spring Boot，而且无需自行管理这些库的版本。。

Spring Boot 就是一些开发好了 maven 模块，只需导入对应模块就能做对应事情.

只需要 springboot 提供了哪些模块，每个模块是干嘛的.最终使用时判断要做那个功能，选择对应模块就 ok.

与 SpringCloud：

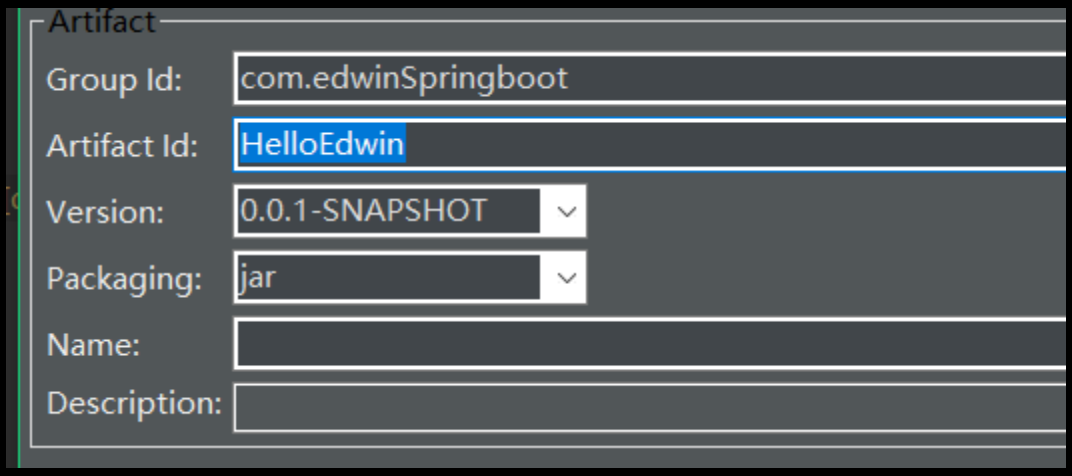
- 微服务框架。
- 微服务通信技术：HTTP+json
- Springcloud 依赖 boot 实现微服务
- Spring boot 不能说是微服务

与 Springmvc

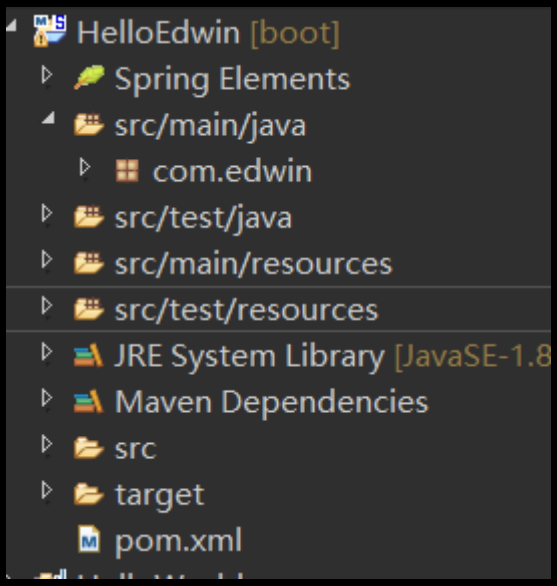
- Boot 的 web 组件集成了 springmvc 框架，但是无启动配置文件

使用步骤

- 创建 Maven 工程：
  - 这里勾选 create a simple projeck(跳过那啥)
  - group ID 和 Artifact ID，
  - Packaging 类型需要选择 jar



完整工作目录：



- 配置 pom.xml

```
<!--spring-boot-starter-parent整合第三方常用框架依赖信息 -->
 <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.1.3.RELEASE</version>
      <relativePath/> <!-- lookup parent from repository -->

 </parent>

  <!--通过dependency引入包依赖
  实现原理：Maven依赖继承关系
      相当于把第三方常用的Maven依赖信息在parent项目中已经封装好了，
      使用springboot提供信息关联整合的jar包
  springboot快速原理：Maven子父依赖关系，springboot对常用依赖信息进行封装
    -->


  <!-- spring-boot-starter-web是springBoot整合SpringMVC -->
   <dependencies>
      <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-web</artifactId>
          <!-- 不需要写版本号，parent中已经封装好了 -->
      </dependency>
   </dependencies>
</project>
```

  - 注意注意：修改后需要 maven---Update Project，不然不会生成 Maven Dependencies 文件夹，同时不会导入 pom.xml 中添加的依赖包，类中也不能使用！总之 pom.xml 一变化就 update!!!
- 编写类并运行（程序写在 src/main/java）

```java
@RestController
//@RestController表示该类中所有的方法返回JSON格式！（@Controller+@ResponseBody）
@EnableAutoConfiguration
//@EnableAutoConfiguration注解作用：
//让Spring Boot根据所声明的依赖来对Spring框架进行自动配置（会自动添加Tomcat、Springmvc）
//注意：扫包范围默认为当前类

//@ComponentScan("com.edwin")
//如果要启动多个：需要使用这个设置扫描包，一般不使用。
public class MemberController {
    //Spring boot 启动原理：Springmvc注解启动，创建内置Tomcat服务器，使用
    //Tomcat加载mvc，启动
    @RequestMapping("/index")
    public String index() {
        return "这个会直接显示在页面上";
    }
    public static void main(String[] args) {
        //整个spring程序的入口，启动spring项目
        SpringApplication.run(MemberController.class, args);
    }
}
```

启动

注意：

- 必须使用@RestController(或 Controller)和 @EnableAutoConfiguration
- @ComponentScan 也可以扫描指定的多个类/包

```java
@ComponentScan(basePackages = { "com.itmayiedu.member.controller", "com.itmayiedu.member.controller"})
```

缺点：不适用于需要扫描的包过多

原理

```java
lic class App {

public static void main(String[] args) {
    //启动springboot应用
    //1)拉起一个内置Tomcat
    //2)也会初始化一个Spring-原来手动配置web.xml,applicationContext-*.xml通通都自动配置
        //里面有一些默认的配置，比如springmvc mybatis 事务配置，等
    //3)会把加了@SpringBootApplication的主键的类当前包，及其子子孙孙包，进行全部注解的扫描，把相关
    //的bean纳入Spring管理
    SpringApplication.run(App.class, args);
}
```

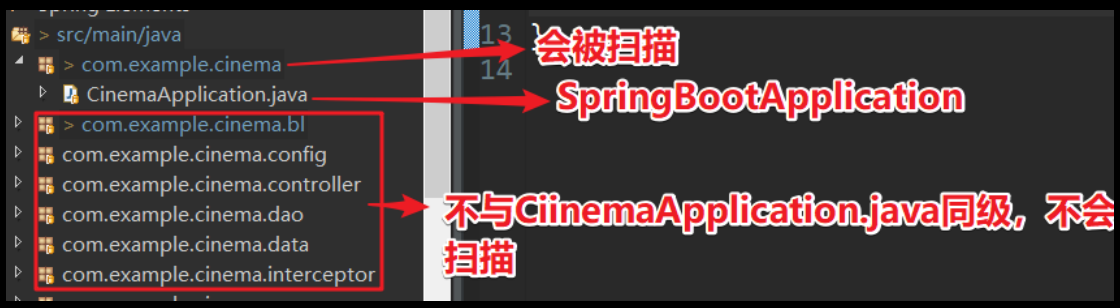
@SpringBootApplication

@SpringBootApplication 注解相当于@EnableAutoConfiguration + @ComponentScan

它会扫描同级包（含子包）和当前包

注意：是注解的那个类所同级的包，不是他所在包的同级包

```java
@SpringBootApplication
public class CinemaApplication {
    public static void main(String[] args) {
        SpringApplication.run(CinemaApplication.class, args);
    }
}
```

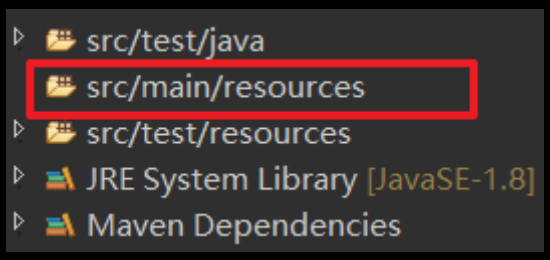


SpringBoot 访问静态资源

SpringBoot 默认提供静态资源目录位置需置于 classpath 下，目录名需要符合以下要求：

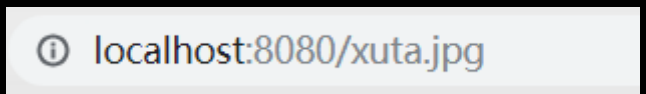
- /static
- /public
- Resources
- /META-INF/resources

我们可以再/src/main/resources/目录下创建 static。



访问：在浏览其 localhost 后面需要加其他目录，只需要使用资源名称即可。即不需要加/static/什么的

因为我们看到的目录结构并不是真正的 URL。



使用 Freemarker 模块引擎渲染 web 视图

Freemarker：做伪静态的 HTML（不使用.jsp）

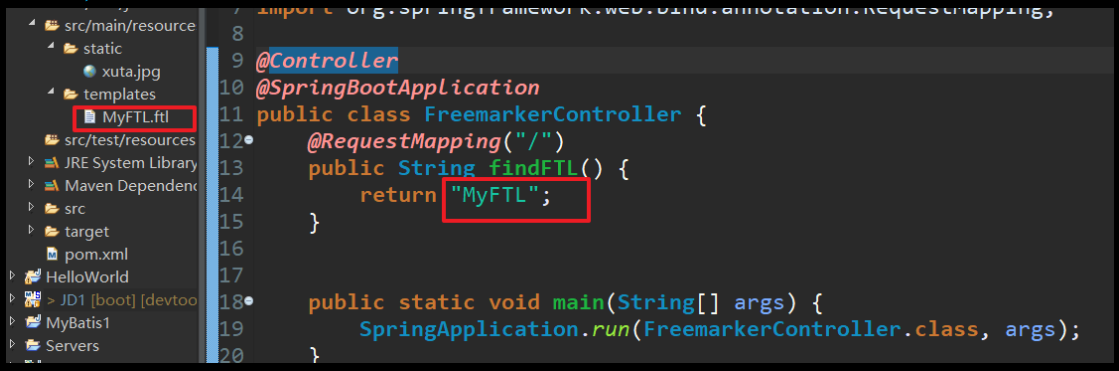
使用步骤：

- 在 pom.xml 中引入：

```xml
<dependencies>
    <!-- 引入freeMarker的依赖包. -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-freemarker</artifactId>
    </dependency>

</dependencies>
```

- 使用，返回文件内容到浏览器



注意：

- Freemaker 的文件后缀：.ftl
- 需要将 ftl 文件放在 src/main/resources/templates 下
- Ftl 可以使用 controller 通过 Map 等传来的参数，格式是：${paramName}

```java
public String findFTL(Map<String, String> map) {
    map.put("name", "edwin");
    map.put("age", "21");
    return "MyFTL";
}
```

```
5 姓名：${name}
6 年龄：${age}
```

FreeMarker 被设计用来生成 HTML Web 页面，特别是基于 MVC（Model View Controller）模式的应用程序。使用 MVC 模式作为动态的 WEB 页面的想法，是为了分隔页面设计者 (HTML 设计者) 和程序员。.每个人做自己擅长的那一部分。设计者可以不通过程序员的改变或修改代码来改变网页的样子，因为应用逻辑(Java 程序)和页面设计（FreeMarker 模版）是分开的。模板不会被复杂繁琐的程序框架所破坏。即使当一个项目的程序员和 HIMTL 页面的制作者是同一个人时，这种分隔也是很有用，因为这样有助于保持应用的清晰并易于维护。

使用 JSP 渲染视图

Springboot 默认不太支持 JSP

不要把 jsp 文件放到 resource 下，可以会被别人访问到；应该放到 webapp/WEB-INF 下

没去自己尝试——第 11 集

全局捕获异常

```
@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
        @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，需要使用
@ResponseBody 进行 json 转换。
                返回 String，表示跳到某个 view。
                返回 modelAndView。
                返回 model + @ResponseBody。
```

全局捕获异常：整个 web 请求项目全局捕获异常
对于一个方法，可能发生异常，必须要处理，不能让用户看到异常信息，以便影响用户体验

```java
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(RuntimeException.class)
    @ResponseBody
    public Map<String, Object> exceptionHandler() {
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("errorCode", "101");
        map.put("errorMsg", "系統错误!");
        return map;
    }
}
```

大公司会将错误写入错误日志
（以后用到再细致学吧，不太懂！）

Log4i 日志

- 新建 log4j 配置文件：log4j.properties
- 添加依赖 pom.xml ：
  <**artifactId**>spring-boot-starter-log4j</**artifactId**>

AOP 统一处理 Web 请求日志

集成 Lombox 让代码更简洁

@Async 异步调用

application.properties

Spring Boot 使用“习惯优于配置”

src/main/resources 下 application.properties 相当于一个默认的配置文件，可以存放许多的键值对，比如 jdbc 数据库的密码用户名等。

在使用的时候用 @Value(“${key}”)进行取用。

注意：文件名不能变，其他的文件名无用，不会被读取

与 springmvc

如果需要使用 jsp 返回渲染，则需要定义 prefix、suffix：

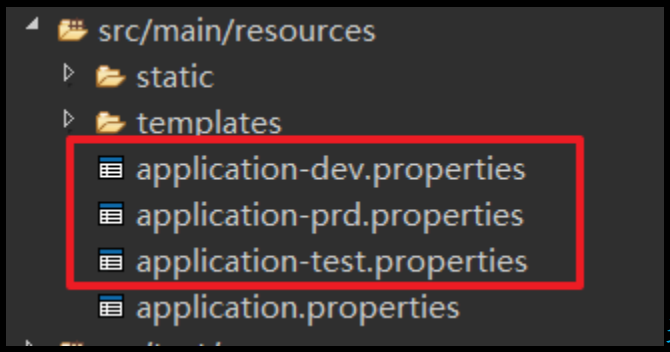
Value 自定义参数

注意：在 src/main/resources 下建立一个文件：application.properties

然后后面可以直接访问这个文件： ${key}

多环境配置

- 本地开发环境：dev
- 测试环境 test
- 预生产环境 pre
- 生产环境：运维方可修改



这三个配置文件在 resources 中声明。

在 application.xml 中使用 spring.profiles.active=prd / test / dev 来指定使用的那个配置文件。

通过指定之后，那么就可以调用另一个文件中的键值对了。

注意：

1. 如果指定了环境，那么三个中的一个和 application.properties 都是可用的。而且，如果二者中的 key 冲突了，以三个中的一个为准!!!

整合 Mybatis

步骤：

- 引入 2 个依赖：pom.xml

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>

<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <!-- 注意：这里必须要加版本！！！ -->
    <version>3.4.6</version>
</dependency>
```

1.mysql依赖

2.mybatis依赖

如果后面不使用 Mapper 而是用 MapperScan 注解，还需要依赖：

```
<!-- 如果需要使用 @MapperScan()注解，则需要导入此包！！ -->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>1.2.0</version>
</dependency>
```

- 配置文件 appplication.properties，添加数据库连接信息

```
spring.datasource.url=jdbc:mysql://localhost/Temp
spring.datasource.username = root
spring.datasource.password = [redacted]
spring.datasource.driver-class = com.mysql.jdbc.Driver
```

（不要使用其他的 key，这个 key 是系统默认的）

- 先建立一个 User 类（和数据库中的表对应）!

```java
public class User {

    private int id;
    private String name;
    private int age;

    public User(String name, int age) {
        super();
//      this.id = id; 这个id让mysql自动添加吧!
        this.name = name;
        this.age = age;
    }
    public User() {}

    public int getId() {
        return id;
    }
```

getter
setter

一般不需要把 id 传入数据库，数据库会按递增自动分配 id

- 编写 DAO 层：UesrMapper.java

使用注解式的 mybatis

```java
package com.myBatis;

import java.util.List;
/**
 * Mapper算DAO。User不算DAO
 */
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Update;

//@Mapper  以后不用这个，但是一定要看得懂！！！
public interface UserMapper {
    @Insert("insert into users(name,age) values(#{name},#{age})")
    public int add(User user) ;

    @Delete("delete from users where id=#{id}")
    public int deleteById(User user);

    @Update("update users set name=#{name},age={#age} where id = #{id}")
    public int update(User user);

    @Select("select * from users where id=#{id}")
    public User getById(int id);

    @Select("select * from users")
    public List<User> getAll();
}
```

- 编写业务层：UserService.java

```java
package com.myBatis;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
/**
 * 如果一个类带了@Service注解，将自动注册到Spring容器，
 * 不需要再在applicationContext.xml文件定义bean了，
 * 类似的还包括@Component、@Repository、@Controller。
 *
 *Service层————业务层
 */
@Service
public class UserService {
    @Autowired   //必须使用自动装配！
    public UserMapper mapper;

    public int insertUser(String name,int age) {
        //这个mapper虽然是一个接口，但是不需要实现，因为它是MyBatis的语句，可以直接使用。
        int insert  = mapper.add(new User(name,age));//这里只使用了插入。
        return insert;
    }

}
```

或者不需要拥有 mapper，实现这个类也是可以的，好像用得更多。

```java
public class UserServiceImpl implements UserMapper{

    @Override
    public User select(User user) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int insert(User user) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int delete(User user) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int update(User user) {
        // TODO Auto-generated method stub
```

- 编写控制层： UserController.java

```java
package com.myBatis;

import org.apache.ibatis.annotations.Param;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
/**
 * 这算是控制层！！
 * 测试boot整合mybatis
 */

@RestController
public class UserController {
    @Autowired  //控制层使用业务层！
    private UserService service;

    @RequestMapping("/index")
    public int operate(@Param("name") String name,@Param("age") int age) {
        return service.insertUser(name,age);
    }
}
```

- 编写程序入口：

```java
package com.myBatis;

import org.mybatis.spring.annotation.MapperScan;

 * mybatis启动方式：

@SpringBootApplication
//@MapperScan("com.myBatis") 通过反射，读取这个包，然后把它装入容器中去。
@MapperScan(basePackages = {"com.myBatis"})
public class MyBatisTestAPP {

    public static void main(String[] args) {
        SpringApplication.run(MyBatisTestAPP.class, args);
    }
}
```

通过Mapper来扫描其他的类，这里启动就把其他层启动了，虽然看不出这与其他次的关系！

Mybatis 启动

* mybatis 启动方式 ：

- 可以再 mapper 层不需要加 mapper 注解，但是一定要在启动类的时候添加 @MapperScan()---------这种方式更好
- 在 mapper 类中使用@Mapper 注解。

- 浏览器访问：localhoat:8080/index?name=Edwin&age21

@Transactional 事务回滚

热部署

Edwin Xu

开发者模式，不用一点小修改都重启 APP

PO VO DTO VO

- VO（View Object）：视图对象，用于展示层，它的作用是把某个指定页面（或组件）的所有数据封装起来。
- DTO（Data Transfer Object）：数据传输对象，这个概念来源于 J2EE 的设计模式，原来的目的是为了 EJB 的分布式应用提供粗粒度的数据实体，以减少分布式调用的次数，从而提高分布式调用的性能和降低网络负载，但在这里，我泛指用于展示层与服务层之间的数据传输对象。
- DO（Domain Object）：领域对象，就是从现实世界中抽象出来的有形或无形的业务实体。
- PO（Persistent Object）：持久化对象，它跟持久层（通常是关系型数据库）的数据结构形成一一对应的映射关系，如果持久层是关系型数据库，那么，数据表中的每个字段（或若干个）就对应 PO 的一个（或若干个）属性。

过程

i. 用户发出请求（可能是填写表单），表单的数据在展示层被匹配为 VO。
ii. 展示层把 VO 转换为服务层对应方法所要求的 DTO，传送给服务层。
iii. 服务层首先根据 DTO 的数据构造（或重建）一个 DO，调用 DO 的业务方法完成具体业务。
iv. 服务层把 DO 转换为持久层对应的 PO（可以使用 ORM 工具，也可以不用），调用持久层的持久化方法，把 PO 传递给它，完成持久化操作。

@RestController

@RestController 注解相当于@ResponseBody ＋ @Controller 合在一起的作用。

1) 如果只是使用@RestController 注解 Controller，则 Controller 中的方法无法返回 jsp 页面，或者 html，配置的视图解析器 InternalResourceViewResolver 不起作用，返回的内容就是 Return 里的内容。

2) 如果需要返回到指定页面，则需要用 @Controller 配合视图解析器 InternalResourceViewResolver 才行。

如果需要返回 JSON，XML 或自定义 mediaType 内容到页面，则需要在对应的方法上加上@ResponseBody 注解。

springboot 常用的 maven 模块

Spring Boot Web

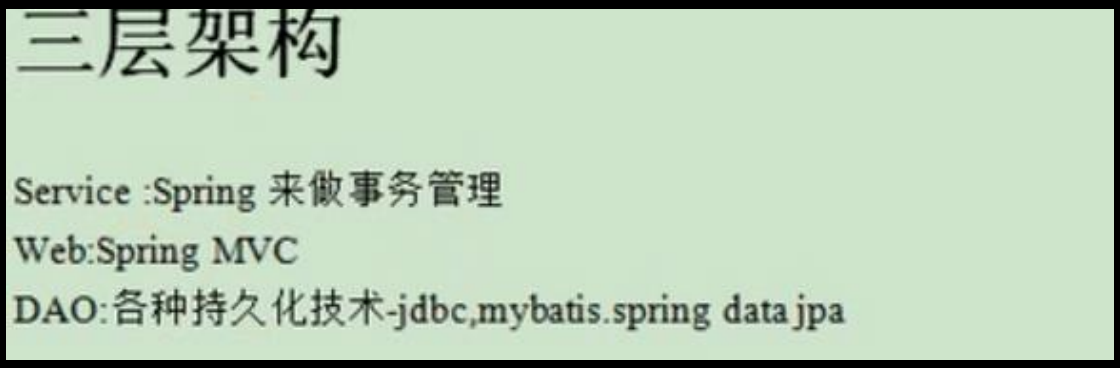
跳转 JSP

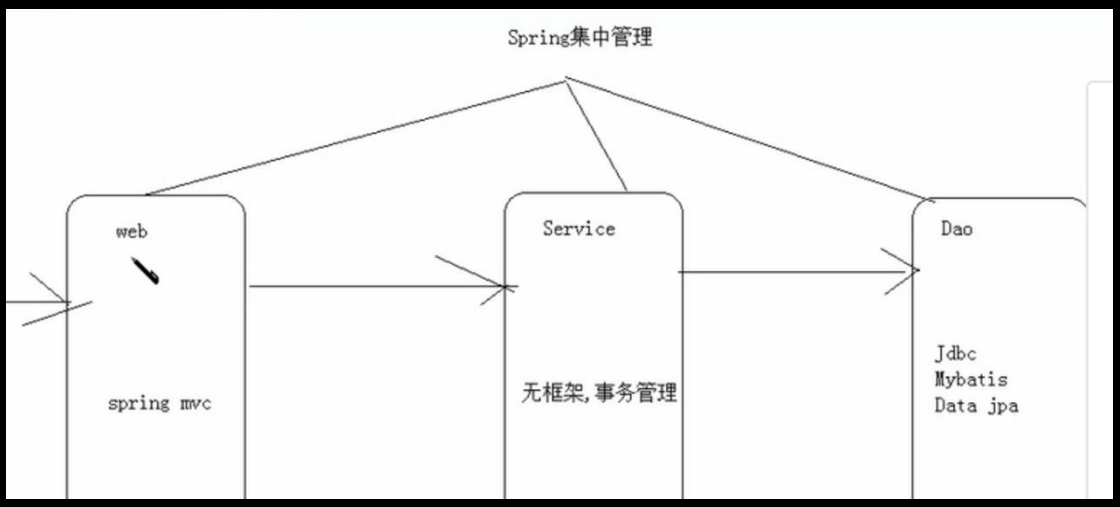
步骤：

- 创建 Maven web project
- 引入依赖
- 配置 application.properties 对 jsp 的支持

```
#this is prefix.
spring.mvc.view.prefix=/WEB-INF/jsp/
#this is suffix.
spring.mvc.view.suffix=.jsp
```

- 编写并启动 springboot 应用
- 编写测试 Controller
  返回页面 jsp
  返回 json





跳转 HTML

首先我们使用 Thymeleaf，必须要先配置（下一个标题）
注意：在 controller 中声明返回的文件名后，系统默认的寻找路径是：Src/main/resources/template，所以返回的这个文件必须位于这个文件内，而且原来的 springmvc.prefix　suffix 对于这个完全没用！

Thymeleaf

Thymeleaf 是一个跟 Velocity、FreeMarker 类似的模板引擎，它可以完全替代 JSP 。

我的 jsp 跑不起来，老师也用的这个，就学这个把。

使用步骤：

- 引入依赖：

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

默认的模板映射路径是：**src/main/resources/templates，**

- 配置 thymeleaf 视图解析器

```
#thymeleaf start
spring.thymeleaf.mode=HTML5
spring.thymeleaf.encoding=UTF-8
spring.thymeleaf.content-type=text/html
#开发时关闭缓存,不然没法看到实时页面
spring.thymeleaf.cache=false
#thymeleaf end
```

- 编写控制器

thymeleaf的使用及配置

```
* th:action    <form id="login" th:action="@{/login}">.....</form>    定义后台控制器的路径
* th:each      循环List集合: <tr th:each="user,iterStat : ${list}"> <td
th:text="${user.userName}">Onions</td> </tr> iterStat:下标
              循环Map集合: <div th:each="mapS:${map}"> <div th:text="${mapS}"></div> </div>
              循环数组:      <div th:each="arrayS:${arrays}"> <div th:text="${arrayS}"></div> </div>
* th:field
* th:href      定义超链接，类似<a>标签的href 属性。value形式为@{/login}
* th:id        类似html标签中的id属性。    <div class="user" th:id = "(${index})"></div>
* th:if        <span th:if="${Sex} == 1" > <input type="redio" name="se" th:value="男" /> </span>
              <span th:if="${Sex} == 2"> <input type="redio" name="se" th:value="女" /> </span>
* th:include
* th:fragment
* th:object
* th:src        外部资源引入    <script th:src="@{/static/js/jquery-2.4.min.js}"></script>
* th:replace
* th:text       <input th:text=${param} />
* th:value      <input th:value=${param} />

条件判断可以这样写：<input th:text="(${user.isAdmin}?'管理员':'普通用户')"></input>
```

```
thymeleaf的配置文件说明
#spring.thymeleaf.cache = true #启用模板缓存。
#spring.thymeleaf.check-template = true #在呈现模板之前检查模板是否存在。
#spring.thymeleaf.check-template-location = true #检查模板位置是否存在。
#spring.thymeleaf.content-type = text / html #Content-Type值。
#spring.thymeleaf.enabled = true #启用MVC Thymeleaf视图分辨率。
#spring.thymeleaf.encoding = UTF-8 #模板编码。
#spring.thymeleaf.excluded-view-names = #应该从解决方案中排除的视图名称的逗号分隔列表。
#spring.thymeleaf.mode = HTML5 #应用于模板的模板模式。另请参见StandardTemplateModeHandlers。
#spring.thymeleaf.prefix = classpath: / templates / #在构建URL时预先查看名称的前缀。
#spring.thymeleaf.suffix = .html #构建URL时附加到查看名称的后缀。
#spring.thymeleaf.template-resolver-order = #链中模板解析器的顺序。
#spring.thymeleaf.view-names = #可以解析的视图名称的逗号分隔列表。/ templates / #在构建URL时先查看名称的
前缀。
#spring.thymeleaf.suffix = .html #构建URL时附加到查看名称的后缀。
#spring.thymeleaf.template-resolver-order = #链中模板解析器的顺序。
#spring.thymeleaf.view-names = #可以解析的视图名称的逗号分隔列表。/ templates / #在构建URL时先查看名称的
前缀。
#spring.thymeleaf.suffix = .html #构建URL时附加到查看名称的后缀。
#spring.thymeleaf.template-resolver-order = #链中模板解析器的顺序。
#spring.thymeleaf.view-names = #可以解析的视图名称的逗号分隔列表。
```

YML

Yml 比 properties 更加节约，简介

```
server.port=80
server.http2.enabled=true
```

```
server:
  port: 80
  http2:
    enabled: true
```

尽量使用 yml

注释：##

注意：

- 每个冒号后面都必须要使用至少一个空格
- 子属性与父属性也必须距离至少一个空格。

邪恶令好人团结

Facing evil brings good people together

只是没有人觉得自己是邪恶的一边

No one ever thinks that they’re the evil one

学习能力：少模仿别人的思想，自己去看源码，去研究，多思考，提高思维能力才是关键。