1. 导入的依赖： </parent>其父依赖是真正的版本仲裁中心
   1. 导入的依赖有如：spring-boot-starter-web
   2. Spring-boot-starter:spring-boot的场景启动器
   3. Spring-boot将各种功能场景都包装成场景启动器。有parent版本控制
2. SpringBootApplication：SpringBoot应用标注在某个雷伤说明这个类是Springboot的主要配置类，Springboot就应该运行这个类的main方法来启动springboot应用
3. @SpringBootConfiguration:Springboot的配置类：
   1. 标注在某个类上，表示这是一个springboot的配置类-------------
   2. 以前我们需要配置的东西，springboot帮我们自动配置；@EnableAutoConfiguration告诉springboot开启自动配置功能，这样自动配置才能生效
   3. @AutoConfigurationPackage：自动配置包
      1. @import(AutoConfigurationPackages.Registar.class)
   4. Spring的底层注解;@import，给容器中导入一个组件，导入的组件由AutoConfigurationPackages.Registar.class将主配置类：@SpringBootApllication标注的类)的所在包及下面所有子包里面的所有组件扫描到Spring容器
   5. @lmport(EnableAutoConfigurationlmportSelector.class) ;

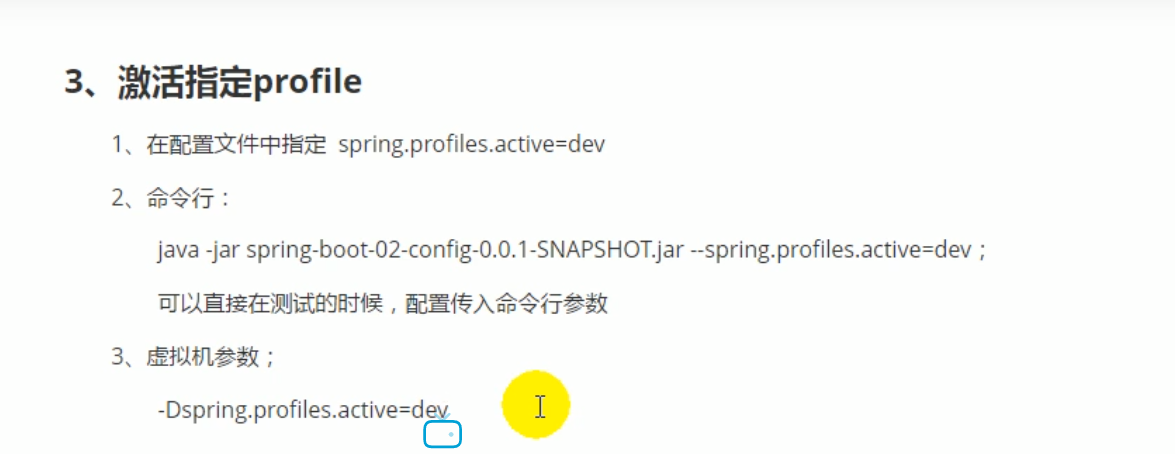
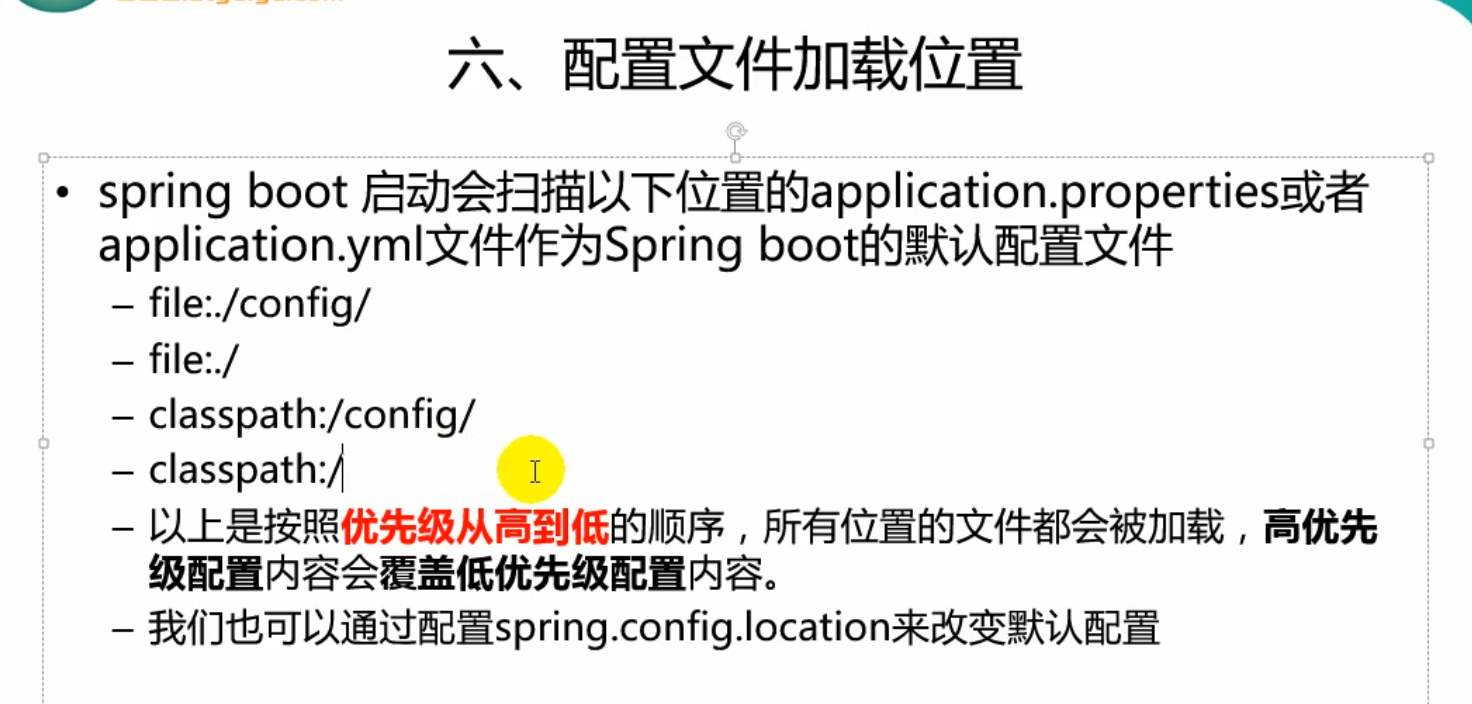
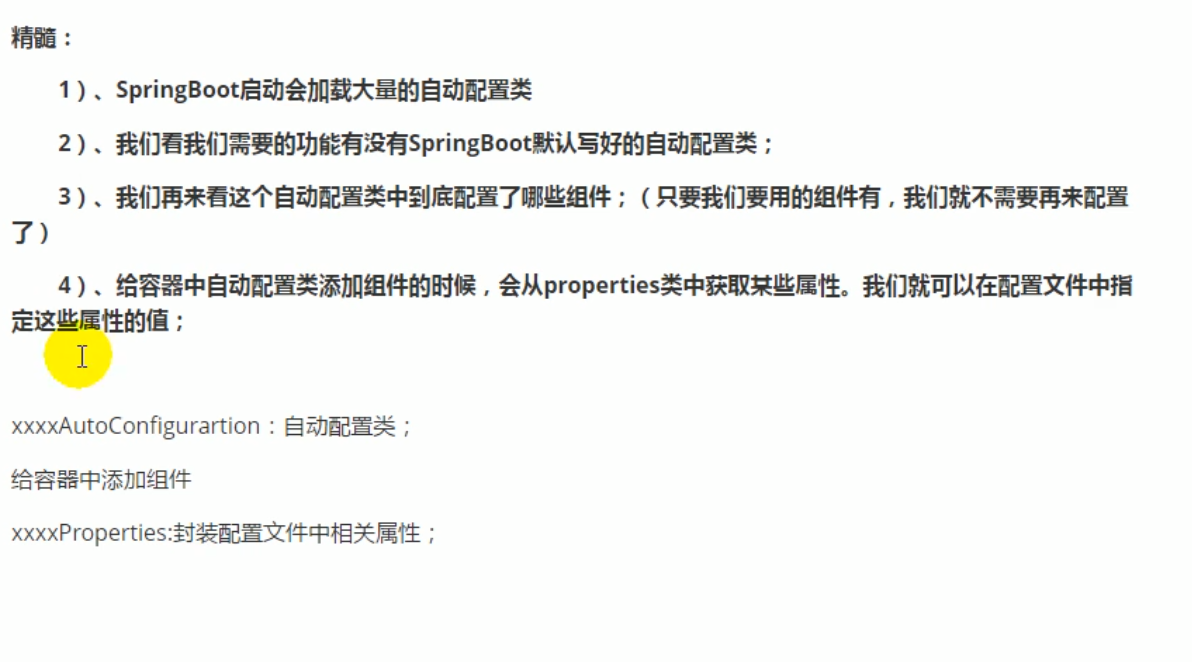
给容器中导入组件? EnableAutoConfigurationlmportSelector:导入哪些组件的选择器;将所有需要导入的组件以全类名的方式返回,这些组件就会添加到容器中

1. Spring Initializr:
   1. 标记语言:以前的配置文件;大多都使用的是XXXx.xml文件;YAML∶以数据为中心，比json、xml等更适合做配置文件;YAML:
   2. 直接输入包名，自动在pom.xml中导入相关parent,plugin,dependency
      1. YML配置文件语法：\*将配置文件中配置的每一个属性的值，映射到这个组件中@ConfigurationProperties（prefix = “person”）:告诉SpringBoot将本类中的所有属性和配置文件中相关的配置进行绑定;,前置进行范围约束
2. 单元测试注意版本,Junit版本需要4.12以上,学会看日志
3. application.properties
   1. 文件注意编码：File Encoding选UTF-8，勾上转ASCII码
   2. 若还有乱码，删除重建
   3. 只要对应类写了  
      @Component  
      @ConfigurationProperties(prefix = "person")

即可使用person.xxx后面不用加空格

1. 配置文件
   1. 可以在person类里通过@value的方式部分修改配置
   2. @Value(“${person.last-name}”)//获取配置文件
   3. @Value{“#{11+7}”}//表达式
   4. @Value{“ture”}//直接赋值
   5. @Value和@ConfigurationProperties区别对比(松散语法可在字母间加\_或-)



* 1. 配置文件占位符：
  2. 当有多个配置文件(profile)时，要在不同的环境下激活不同的配置文件（spring.profiles.active=dev）：
  3. 配置加载顺序：
  4. SpringBoot的自动配置原理：

1. 日志：使用slf4j包
   1. // Person person = new Person("zhang","f",25);  
       //日志记录器  
       Logger logger = LoggerFactory.getLogger(getClass());  
       @Test  
       public void contextLoads(){  
       logger.trace("这是trace日志");  
       logger.debug("这是debug日志");  
       logger.info("这是info日志");  
       logger.warn("这是warn日志");  
       logger.error("这是erro日志");
      1. 在application.propoties配置中设置logging.level.com.dell=trace//设置默认级别
2. 资源：
   1. Public 公共资源，可直接访问
   2. Templates Thymleaf放置
3. Web开发
   1. 
   2. Thymleaf版本切换 <themeleaf.version>3.0.2.RELEASE</themeleaf.version>

<thymeleaf-layout-dialect.version>2.2.2</thymeleaf-layout-dialect.version>

* 1. Thymleaf检查严格，注意<meta/>标签(所有标签！！！)后面跟反斜杠
  2. Thymleaf导入语法提示：<html lang="en" xmlns:th="http://www.thymeleaf.org">|
  3. Thymleaf常用语法：
     1. th:text=”${name}”//转义，将该值以字符串赋值到对应标签
     2. th:utext//不转义,显示标签,常用
     3. th:each(“user:${users}”)遍历循环
     4. th:inline(“text”)在其子类下可以使用内联框架改为javascript可在js中使用

1. SpringMVC配置拓展
   1. @Configuration

public class configClass extends WebMvcConfigurerAdapter {}

在其中重写各种方法：如视图解析器

@Override

public void addViewControllers(ViewControllerRegistry registry) {

//视图解析器跳转

registry.addViewController("/tiaozhuan").setViewName("success");

}

* 1. 启用@EnableWebMvc将全面接管SpringMVC，springboot对mvc的自动配置都不要了（不要写）
  2. 国际化资源配置巨坑！！
     1. spring.messages.basename=i18n/login，现版本不兼容用点的方式
     2. 要有一个默认的xxx,properties文件
  3. 更改地区请求头(理论，未实现)
     1. implements LocaleResolver
     2. @Override

public Locale resolveLocale(HttpServletRequest request) {

locale = new Locale(split[0],split[1]);

return locale;

* 1. 实际开发中注意：
     1. 禁用视图缓存：spring.thymeleaf.cache=false
     2. Ctrl+F9重新Build项目，即可实现类似热部署

1. 控制器类接收请求并处理：
   1. 控制器类加@Controller
   2. @PostMapping(value = {"/user/login"})/\*不同的请求方式调用不同的Mapping方便点\*/
   3. 注解方法里写入参数，作为接收，如：@RequestParam("username") String userName,

@RequestParam("userpass") String userPass,

Map<String,Object> map

* 1. 通过map传递参数回去(为什么一定要用map?)
  2. 不能放在th:text里的数据用[[来存放]]
  3. Form表单action的地址：th:action="@{/user/login}",与mapping里的value对应，记得加上submit按钮
  4. 在html页面中进行逻辑判断：th:if(&not #strings.isEmpty(msg))此优先级优先于th:text所以判断决定后者是否执行
  5. @ ResponseBody将返回的界面解析成真正的字符串，，，，有什么用…

1. 拦截器
   1. 实现HandlerInterceptor接口，实现重写方法1prehenlder:通过session判断是否用户存入到session中，若session为空则传参至login界面,通过request的转发方法

：httpServletRequest.getRequestDispatcher("/login").forward(httpServletRequest,httpServletResponse);跳转回最初的界面然后return false;否则return true;

* 1. 在配置类中重写addInterceptors方法，拦截所有请求除了登录界面，及登录跳转验证界面

registry.addInterceptor(new LoginHandlerInterceptor()).addPathPatterns("/\*\*")

.excludePathPatterns("/login","/","/user/login");//排除掉

1. Thymeleaf
   1. 公共片段引用：
      1. th:fragment=”name”//利用片段名插入
      2. th:insert/replace/include //插入，替换，包含
         1. th:replace=”~{html::name}”//利用片段名
         2. th:replace=”~{html::#id}”//利用id值插入
      3. 点击选项卡高亮逻辑：
         1. 在每个界面引入模板时传入参数如：th:replace=”commons/bar::#sidebar(activeUri=’main.html’)”//引入commons目录下bar文件的id为sidebar的模板，参数自定义，用于判断是否该高亮
         2. 得到参数后怎么判断：th:class=”${activeUri==’heighLight’?’nav-link active’:’nav-link’}”三元赋值class
   2. 从Dao层引入数据进入列表
      1. Date数据用SimpleDateFormat来创建，淘汰的方法有BUG
      2. 在控制器类中，将Dao层的数据利用@ Autowired注入(加上@ Component注解)后，调用dao层类的返回所有实体类的方法得到所有实体，将他们转成Collection集合，用mode.setattibute返回到前台
      3. 前台用th:each接收，得到各种数据，时间类用#dates.format解析
   3. 从页面传输数据回controller类，
      1. Controller类在requestmapping中接收添加请求(/addPage)
      2. 查出所有的需要选择的数据，打包发送到前台提供选择
      3. 单选用input标签 radio类型 name值相同 value值不同
      4. 下拉框用select标签，<option th:each(“dep:${departments}”)***自动封装有问题，无法自动封装自定义类，选择手动封装***
   4. 从前台传输回控制层
      1. 在Contoller类中使用postmapping接收请求(/addPage)，因为请求方式不同，所以不会冲突
      2. 获取参数，封装好user类，利用@Autowired注入dao类，添加到该dao类中，最后用跳转或转发触发controller拦截请求，最后显示dao类的所有员工
      3. 因为缓存问题，js一旦被引入使用就会被缓存，此时修改js的代码将不再生效，只能将js文件名换掉,重新引入
   5. 页面编辑：
      1. input使用th:value(${emp.getid})
      2. radio使用th:checked(${emp.getGender==1})
      3. option使用th:selected(${emp.getDepId}==${dep.getDepid}?ture:false)
      4. date使用th:value = ${emp.getStringDate()}//日期转1999-06-03字符
      5. 在图片传值时要加上/如：th:style=”’background-image:’+’url(/img/loginBack.jpg)’”
      6. 在进行重定向和转发时请也加上/，这是个好习惯，编译器只在运气好时会帮你修复你的语法错误
   6. 删除员工
      1. 我的思路：[a标签直接@{/delStudent/}+${stu.getId()}](mailto:a标签直接@%7b/delStudent/%7d+$%7bstu.getId()%7d),拿到之后删除跳转：return

“redirect:/studentList”;

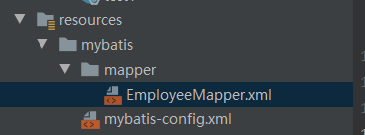
* + 1. 老师思路：
       1. 去除a标签，在每个按钮上加上一个自定义属性：利用th:attr来添加自定义属性：th:attr = “myClass=[@{/delStudent/}+${stu.getId()}](mailto:a标签直接@%7b/delStudent/%7d+$%7bstu.getId()%7d)”：
       2. 定义一个form隐藏的表单，在js代码中给每一个button添加监听，点击之后获取该按钮的自定义属性替换掉form表单的action地址，并提交form表单。，，，，，，，这和使用a标签区别有多大呢、、、、？
       3. Attr其实是原生getattribute的jquery简化版

1. Docket容器
   1. 将mysql，tomcat，等等运行环境包装到docket容器中，以后只需要启动封装好的docket容器即可生效
   2. 安装虚拟机，virtuialBox 使用一个客户端连接Linux服务器
   3. Linux命名：
      1. 刚配好虚拟机对yum进行换源：https://blog.csdn.net/qq\_25760623/article/details/88657491
      2. Ip attr 查看虚拟机IP地址
      3. Cd /etc/docker 进入目录
      4. Vi /etc/docker/daemon.json 按i进入编辑，按esc输入:wq退出保存
      5. Tee /etc/docker/daemon.json <<-“EOF” 输入内容 输入如EOF保存退出
      6. Systemtcl start docker 启动docker
      7. Systemtcl restart docker重启docker
      8. Systemtcl daemon-reload重新载入镜像源
      9. Yum install -y docker-ce 安装docker
      10. Docker search mysql 搜索容器
      11. Docker pull mysql 拉取mysql容器
      12. Docker images 查看所有拉取的镜像
      13. Docker run –name mytomcat -d tomcat:latest 利用镜像启动tomcat容器
      14. Docker start 容器的id：····~~启动一个存在的容器
      15. Docker stop 容器的id：一长串的字符
      16. Docker ps查看启动的容器，加上-a参数查看所有启动过的容器
      17. Docker rm 容器的id：~~
      18. Docker run -p 3306:3306 –name mysql01 -e MYSQL\_ROOT\_PASSWORD=mysql密码 -d mysql mysql容器的启动需要一个密码，根据官方文档操作
      19. 通过Navicat对mysql进行链接：
          1. 进入mysql容器:docker exec -it mysql02 bash
          2. 对远程链接进行授权:GRANT ALL ON \*.\* TO ‘ROOT’@’%’
          3. 更改密码的加密规则：ALTER USER 'root'@'%' IDENTIFIED BY 'password' PASSWORD EXPIRE NEVER;
          4. ALTER USER 'root'@'%' IDENTIFIED WITH mysql\_native\_password BY 'zhengchuang';
          5. Flush privileges;
          6. 链接地址是虚拟机的地址：ip attr
2. 数据库连接
   1. 配置注意
      1. ?useSSL=false ssl检查报错添加
      2. 在配置Druid数据源时yml文件里将filter改为：druid.filters: stat,wall,log4j2
      3. url: url: jdbc:mysql://10.69.0.244:3306/数据库名?useSSL=false
      4. 3306端口其实是虚拟机的3306端口映射出来的
   2. 流程
      1. 定义interface 使用mapper注解
      2. 设计方法名以及传入的参数,再根据不同的方法搭配不同的注解,这类注解可以读取自定义对象内的属性：
         1. :@Select(“select \* from department where id=#{id}”)
         2. @Delete(“delete from department where id=#{id}”)
         3. 插入时为了匹配自增：
            1. @Option(userGeneratedKeys = true,keyProperty = “id”)
            2. @Insert(“insert into department(department\_name) values(#{departmnetName})”)
         4. @Update(“update department set departmentName=#{departmentName} where id=#{id}”)
      3. 定义controller接收请求调用mapper
         1. 注解使用RestController不返回界面，直接打印json数据
         2. Mapping小解：
            1. Getmapping 处理get请求
            2. Postmapping 处理post请求
            3. Requestmapping两者都能处理，但不能被以上两个代替，相当于父类
         3. 当执行插入操作时：@GetMapping(value = “/dep”)

Public Department insert(Department department){}

此时只需要在路劲后面根?附带上dep类的某个属性，就会自动封装成该类：<http://localhost:8080/dep?departmentName=C3>

因为id是自增的，所以在mapper里加入@Option注解：@Options(useGeneratedKeys = true,keyProperty = "id")

* + - 1. 因为查询语句过多，mapper为了分类可能要写好多个，所以省略在mapper接口上的@Mapper注解，替换成在主函数上写@MapperScan(value=”com.dell.springboot.mapper”)
  1. 用配置文件做CRUD
     1. 目录结构：
     2. 注意一定要加.xml，不然在yml中识别不到
     3. Config中设置各种特殊设置需求，在mapper中写sql语句
        1. <mapper namespace="com.dell.springboot.mapper.EmployeeMapper">
        2. Sql语句可以像注解那样接收#{id}参数
        3. Sql语句注意和数据库属性名对应，不要把自己坑了
     4. 在YML中配置两种xml文件：

mybatis:  
 config-location: classpath:mybatis/mybatis-config.xml  
 mapper-locations: classpath:mybatis/mapper/\*.xml