首先，创建一个maven项目

1.在pom.xml文件中引入项目依赖的jar包，在dependencies标签里面引入（在maven的中央仓库里找，冒号后面的名字是在中央仓库输入以方便查找的名字）

1）springmvc模块：spring web mvc

2）spring的事务控制模块：spring jdbc

3）spring的面向切面编程：spring Aspects

4）Mybatis：Mybatis

5）Mybatis-spring适配包：搜出来Mybatis，第二个就是

6）Mysql驱动

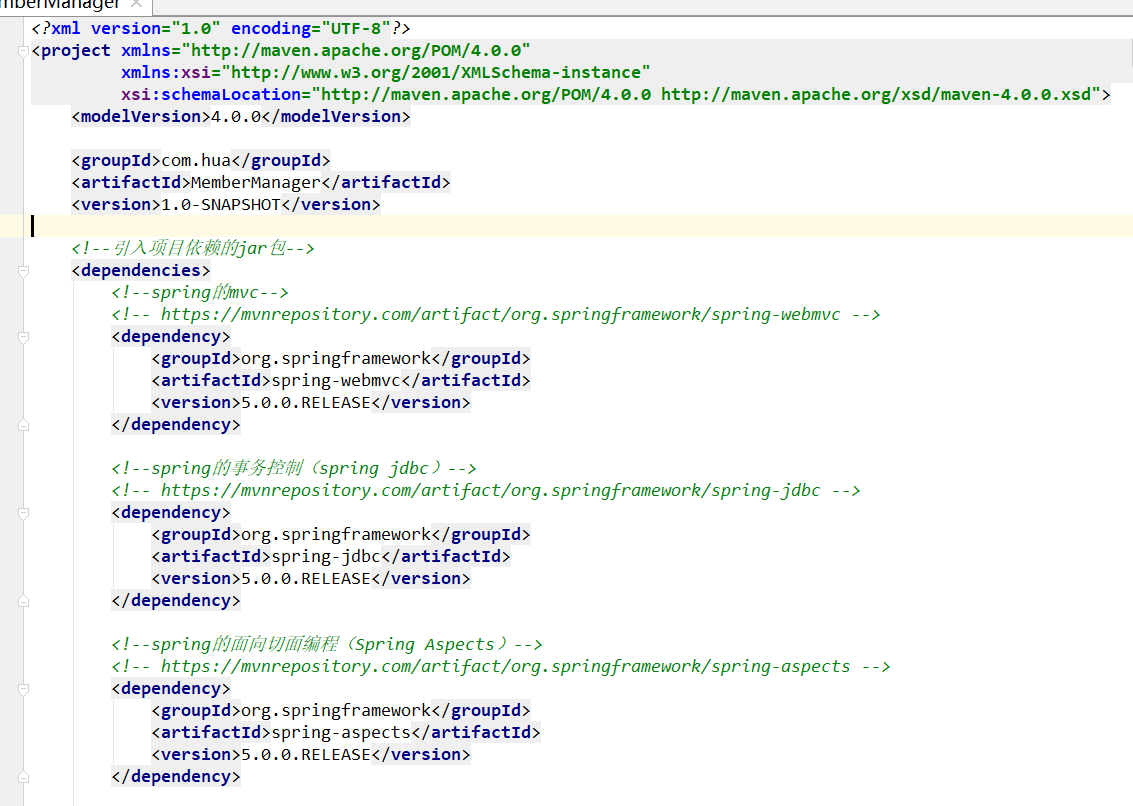
7）c3p0连接池：c3p0

8）jstl：jstl

9）servlet-api：servlet。这个包是服务器上就有的，当项目发布到服务器上的时候，会把这个包剔除掉，所以要有个scope标签，内容为provided，这样才能剔除（最好找到3.0.x版本，因为后面做spring测试的时候需要这个版本的支持）

10）junit：junit

11）jsp，，，，这个我竟然忘记了







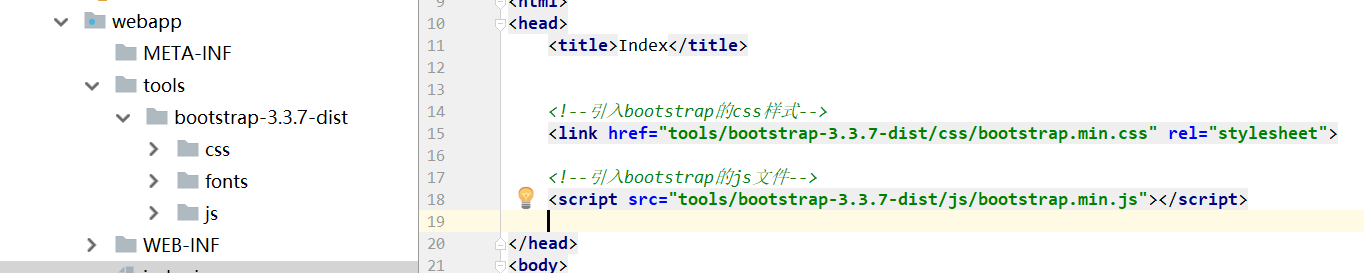
2.引入bootstrap前端框架

1）搜索bootstrap中文网，下载，bootstrap



2）下载完之后，解压完，将文件拷入项目中

3）如果想在页面中使用这些个样式的话，就在头部引入css和js



4）想引入的样式去官网上找吧！



3.编写web.xml的配置文件

1）配置spring容器，让项目一启动，即加载spring的配置文件

2）配置springmvc的前端拦截器，拦截所有请求

3）配置编码过滤器

4）如果项目要用到REST风格URI，则需配置HiddenHttpMethodFilter

5）后期需要什么配置再加

*<?***xml version="1.0" encoding="UTF-8"***?>*<**web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  
 version="3.1"**>  
  
 *<!--启动spring的容器-->* <**context-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:applicationContext.xml</**param-value**>  
 </**context-param**>  
 *<!--这个配置：以让项目一启动，加载某一处的spring的配置文件（差不多是这意思）-->* <**listener**>  
 <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  
 </**listener**>  
  
 *<!--springmvc的前端控制器，拦截所有请求-->* <**servlet**>  
 <**servlet-name**>springDispatcherServlet</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
 *<!--如果不指定，name就要和serlet这个名相同并且目录与web.xml同一级别-->* <**init-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:springmvc-servlet.xml</**param-value**>  
 </**init-param**>  
 </**servlet**>  
  
 <**servlet-mapping**>  
 <**servlet-name**>springDispatcherServlet</**servlet-name**>  
 <**url-pattern**>/</**url-pattern**>  
 </**servlet-mapping**>  
  
  
 *<!--字符编码过滤器-->* <**filter**>  
 <**filter-name**>characterEncodingFilter</**filter-name**>  
 <**filter-class**>org.springframework.web.filter.CharacterEncodingFilter</**filter-class**>  
  
 <**init-param**>  
 <**param-name**>encoding</**param-name**>  
 <**param-value**>UTF-8</**param-value**>  
 </**init-param**>  
  
 <**init-param**>  
 <**param-name**>forceRequestEncoding</**param-name**>  
 <**param-value**>true</**param-value**>  
 </**init-param**>  
  
 <**init-param**>  
 <**param-name**>forceResponseEncoding</**param-name**>  
 <**param-value**>true</**param-value**>  
 </**init-param**>  
 </**filter**>  
 <**filter-mapping**>  
 <**filter-name**>characterEncodingFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
 </**filter-mapping**>  
 *<!--REST，将POST请求转换为PUT请求和DELETE请求的过滤器，注意多个过滤器要把字符编码过滤器的mapping放在前面-->* <**filter**>  
 <**filter-name**>hiddenHttpMethodFilter</**filter-name**>  
 <**filter-class**>org.springframework.web.filter.HiddenHttpMethodFilter</**filter-class**>  
 </**filter**>  
 <**filter-mapping**>  
 <**filter-name**>hiddenHttpMethodFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
 </**filter-mapping**>  
  
</**web-app**>

4.编写springmvc的配置文件

1）配置springmvc自动扫描的包（此处配置只让它扫描controller组件，这样可以防止在spring和springmvc配置文件中把所有的组件都扫描一遍）

2）配置视图解析器

3）两个标准配置（一般都要进行配置的配置）

4）基本配置如上，后面需要什么配置再进行添加

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>  
  
 *<!--springmvc的配置文件，包含网站跳转逻辑的控制，配置-->  
  
 <!--配置自动扫描的包-->* <**context:component-scan base-package="com.hua" use-default-filters="false"**>  
 *<!--配置只扫描控制器-->* <**context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"**/>  
 </**context:component-scan**>  
  
 *<!--配置视图解析器，方便页面的返回-->* <**bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  
 <**property name="prefix" value="/WEB-INF/views/"**/>  
 <**property name="suffix" value=".jsp"**/>  
 </**bean**>  
  
 *<!--两个标准配置-->  
 <!--将springmvc不能处理的web资源交给tomcat-->* <**mvc:default-servlet-handler**/>  
 *<!--这个注解功能很多，jsr303，快捷ajax，映射动态请求-->* <**mvc:annotation-driven**/>  
  
  
</**beans**>  
  
5.编写db.properties（编写数据库连接的相关信息）

1）url

2）driverClass

3）user

4）password

5)数据库连接池的相关配置（需要的话配置一下），都配置数据相关信息了，你也得有相关的数据库和表了，所以说你要建立一个数据库和一些表，自己根据实际情况来了

**jdbc.jdbcUrl**=**jdbc:mysql://localhost:3306/ssm  
jdbc.driverClass**=**com.mysql.jdbc.Driver  
jdbc.user**=**root  
jdbc.password**=**171911**

6.编写spring配置文件

1）配置spring自动扫描的包（此处和上面springmvc的自动扫描的包如此配置原因相同）

2）配置数据库连接的相关信息（在此之前导入引用外部文件db.properties）

3）配置事务控制（spring中的声明式事务）

4）配置spring和mybatis的整合（mybatis中的SqlSessionFactory）

5）配置扫描器（扫描dao接口）

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd"**>  
  
 *<!--Spring的配置文件。这里主要配置和业务逻辑有关的-->  
  
 <!--配置自动扫描的包,配置spring文件中的自动扫描不扫描控制器-->* <**context:component-scan base-package="com.hua"**>  
 <**context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"**/>  
 </**context:component-scan**>  
  
  
  
 *<!--引入外部数据库配置文件-->* <**context:property-placeholder location="classpath:dbconfig.properties"**/>  
  
 *<!--数据源-->* <**bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>  
 <**property name="jdbcUrl" value="${jdbc.jdbcUrl}"**/>  
 <**property name="driverClass" value="${jdbc.driverClass}"**/>  
 <**property name="user" value="${jdbc.user}"**/>  
 <**property name="password" value="${jdbc.password}"**/>  
 </**bean**>  
  
 *<!--事务控制-->* <**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  
 *<!--控制住数据源-->* <**property name="dataSource" ref="dataSource"**/>  
 </**bean**>  
  
 *<!--事务控制的配置-->* <**aop:config**>  
 *<!--切入点表达式-->* <**aop:pointcut id="txPoint" expression="execution(\* com.hua.service..\*(..))"**/>  
 *<!--配置事务增强-->* <**aop:advisor advice-ref="txAdvice" pointcut-ref="txPoint"**/>  
 </**aop:config**>  
  
 *<!--配置事务增强，事务如何切入-->* <**tx:advice id="txAdvice" transaction-manager="transactionManager"**>  
 <**tx:attributes**>  
 *<!--所有方法都是事务方法-->* <**tx:method name="get\*" read-only="true"**/>  
 <**tx:method name="\*"**/>  
 </**tx:attributes**>  
 </**tx:advice**>  
  
 *<!--配置和mybatis的整合-->* <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  
 *<!--指定mybatis全局配置文件的位置-->* <**property name="configLocation" value="classpath:mybatis-config.xml"**/>  
  
 *<!--指定mybatis的mapper配置文件的位置-->* <**property name="mapperLocations" value="classpath:mapper/\*.xml"**/>  
 *<!--指定数据源-->* <**property name="dataSource" ref="dataSource"**/>  
 </**bean**>  
  
 *<!--配置扫描器，将mybatis的dao接口的实现加入到IOC容器中-->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  
 *<!--扫描所有的dao接口，加入到IOC容器中-->* <**property name="basePackage" value="com.hua.dao"**/>  
 </**bean**>  
  
</**beans**>

7.编写mybatis的配置文件（不写这个配置文件也可以，把所有想用到的配置写到spring里面也行，但是一般都写一个这样的配置文件）

1）自己爱配什么配点什么

2）我喜欢的配置，驼峰命名规则和数据中的在中间加一条下划线的命名规则对应

3）我喜欢的配置，给所有实体类配置别名

*<?***xml version="1.0" encoding="UTF-8"***?>***<!DOCTYPE configuration  
 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-config.dtd"*>***<**configuration**>  
 *<!--开启驼峰命名规则-->* <**settings**>  
 <**setting name="mapUnderscoreToCamelCase" value="true"**/>  
 </**settings**>  
  
 *<!--别名的配置-->* <**typeAliases** >  
 <**package name="com.hua.entities"**/>  
 </**typeAliases**>  
</**configuration**>

8.此处使用了mybatis逆向工程来生成mapper还有实体类还有dao接口（参见mybatis逆向工程文档）

9.生成完mapper文件之后，会发现其中没有级联查询的方法，可以在其中手动添加级联查询的方法

10.测试一下生成的mapper好用吗，测试完所有的方法，spring的项目推荐使用spring的单元测试类

1）.首先引入spring测试的jar包

<**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-test</**artifactId**>  
 <**version**>4.3.7.RELEASE</**version**>  
 <**scope**>test</**scope**>  
</**dependency**>

2）.其次要使用@ContextConfiguration指定spring配置文件位置

3）.然后用@RunWith指定使用什么类型的单元测试类

4）.然后在其中想使用什么组件就使用什么组件，用@Autowired注解标注就行

@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations = {**"classpath:applicationContext.xml"**})  
**public class** MapperTest {  
 */\*\*  
 \* 测试Department的Mapper  
 \* 推荐spring的项目就可以使用spring的单元测试  
 \* 1.导入spring的测试模块  
 \* 2.@ContextConfiguration指定spring配置文件位置  
 \* 3.@RunWith指定使用什么类型的单元测试类  
 \* 4.直接使用@Autowired注解来标注  
 \*/* @Autowired  
 DepartmentMapper **departmentMapper**;  
  
 @Autowired  
 EmployeeMapper **employeeMapper**;  
  
 @Autowired  
 SqlSession **sqlSession**;  
 @Test  
 **public void** testCRUD(){  
 *//1.测试部门的插入  
 // departmentMapper.insertSelective(new Department(null,"开发部"));  
 // departmentMapper.insertSelective(new Department(null,"测试部"));  
 //2.生成员工数据，员工插入  
 //employeeMapper.insertSelective(new Employee(null,"化红磊","M","1719112539qq.com",1));  
 //3.批量插入多个员工，使用uuid来定义员工名，可以执行批量操作的Sqlsession  
 /\*  
 EmployeeMapper employeeMapper= sqlSession.getMapper(EmployeeMapper.class);  
 for (int i = 0; i < 1000; i++) {  
 String uuid= UUID.randomUUID().toString().substring(0,5);  
 employeeMapper.insertSelective(new Employee(null, uuid,"M",uuid+"@qq.com",2));  
 }  
 \*/* }  
}

这个东西吧，这里为了能一次性插入多条数据，使用一个能批量操作的Sqlsession

*<!--方便以后使用，配置一个可以执行批量的sqlSession-->*<**bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate"**>  
 <**constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory"**/>  
 <**constructor-arg name="executorType" value="BATCH"**/>

11.测试完dao层之后，就可以写增删改查了，首先项目一启动进入的页面就是一个列表查询，所以直接查这么多数据，需要进行那啥子，那个肯定一页肯定显示不完，所以此处的知识点是分页查询

1）访问index.jsp页面（URI：/emps）

2）index.jsp页面发送出查询员工列表请求

3）EmployeeController来接受请求

4）来到list页面进行展示

@RequestMapping(**"/emps"**)  
**public** String getEmps(@RequestParam(value = **"pageNumber"**,defaultValue = **"1"**) Integer pageNumber,Model model){  
 *//这不是一个分页查询  
 //引用PageHelper分页插件  
 //在查询之前只需要调用如下方法，并且指定从第几页开始查，还有要写上每页有多少条数据  
 //这样的话startPage方法紧跟的查询就是分页查询* PageHelper.*startPage*(pageNumber,5);  
 List<Employee> emps=**employeeService**.getAll();  
 *//查出来之后还可以用PageInfo对结果进行包装,所以说只需要将PageInfo交给页面即可  
 //封装了详细的分页信息，包括我们查询出来的数据,PageInfo还有个构造器，可以传入可以显示的页数  
 //PageInfo page = new PageInfo(emps);* PageInfo page=**new** PageInfo(emps,5);  
 model.addAttribute(**"pageInfo"**,page);  
 **return "list"**;  
}

12.测试controller

跟上面测试dao层一样，还是用spring的测试

1）引入jar（上边已经引入）

2）.其次要使用@ContextConfiguration指定spring配置文件位置

3）.然后用@RunWith指定使用什么类型的单元测试类

4）[.然后用@WebAppConfiguration](mailto:.然后用@WebAppConfiguration)来标注要使用mvc的ioc容器（下面要用）

5）然后用什么组件用@Autowire标注

6）模拟请求，拿到返回值

7）请求成功之后，请求域中会有pageInfo，我们可以取出pageInfo进行验证，所以这一步骤为取出pageInfo

8）将pageInfo中的内容打印出来进行验证

*/\*\*  
 \* 使用spring模块的测试请求功能，测试CRUD请求的正确性  
 \*/*@RunWith(SpringJUnit4ClassRunner.**class**)  
@WebAppConfiguration  
@ContextConfiguration(locations = {**"classpath:applicationContext.xml"**,**"classpath:springmvc-servlet.xml"**})  
**public class** MvcTest {  
 *//传入springmvc的ioc* @Autowired  
 WebApplicationContext **context**;  
  
 *//虚拟mvc请求，获取到处理结果* MockMvc **mockMvc**;  
  
 @Before  
 **public void** initMockMvc(){  
 **mockMvc**=MockMvcBuilders.*webAppContextSetup*(**context**).build();  
 }  
 @Test  
 **public void** testPage() **throws** Exception{  
 *//模拟请求，拿到返回值* MvcResult result= **mockMvc**.perform(MockMvcRequestBuilders.*get*(**"/emps"**).param(**"pageNumber"**,**"1"**))  
 .andReturn();  
 *//请求成功以后，请求域中会有pageInfo，我们可以取出pageInfo进行验证* MockHttpServletRequest request= result.getRequest();  
 PageInfo pageInfo =(PageInfo) request.getAttribute(**"pageInfo"**);  
 *//获取页码信息* System.***out***.println(**"当前页码"**+pageInfo.getPageNum());  
 System.***out***.println(**"总页码"**+pageInfo.getPages());  
 System.***out***.println(**"总记录数"**+pageInfo.getTotal());  
 **int**[] nums=pageInfo.getNavigatepageNums();  
 **for** (**int** num : nums) {  
 System.***out***.print(**" "**+num);  
 }  
 *//获取员工数据* List<Employee> list= pageInfo.getList();  
 **for** (Employee employee : list) {  
 System.***out***.println(employee.getEmpId()+**"==>"**+employee.getEmpName());  
 }  
 }  
  
}

测试期间会发现其中会报错，因为前面导入的servlet包是2.5.x，比较老的版本，去maven仓库中找到3.0.0版本，配置其中

13.写list界面（前段样式部分，后端代码部分后面写）（即使用bootstrap来写，这里主要记录如何从官方文档中找到相应的部分）

0）.首先，什么事都不干也得导入上面所说过的bootstrap对应的css和js还有jquery，注意，此处以服务器路径为基准（/项目名），而此处项目名也不是写死的，需要进行拼接

*<%--以/开始的相对路径，找资源，以服务器路径为标准--%>***<%** pageContext.setAttribute(**"APP\_PATH"**,request.getContextPath());  
**%>***<!--引入bootstrap的css样式-->*<**link href="${**APP\_PATH**}/webapp/tools/bootstrap-3.3.7-dist/css/bootstrap.min.css" rel="stylesheet"**>  
  
*<!--引入bootstrap的js文件-->*<**script src="${**APP\_PATH**}/webapp/tools/bootstrap-3.3.7-dist/js/bootstrap.min.js"**></**script**>  
  
*<!--引入jquery-->*<**script type="text/javascript" src="${**APP\_PATH**}/webapp/tools/jquery/jquery-3.3.1.js"**></**script**>

1）页面布局（bootstrap中的栅格系统）：此处有标题、（添加删除）按钮、显示的数据表、页码信息

所以说整个行分为四行

所有页面信息都写在<div class=”container”></div>里面

在里面的一行的信息都写在<div class=”row”></div>里面

在一行里面的某一列的信写在<div class="col-md-12"></div>里面，其中那个col指column，md是指大小，不同大小看文档，12是指占据几个列，一共有12个列，可以用这个设置占据几列，如果还想设置偏移的量（比如说设置在最右边就要设置偏移的量）<div class="col-md-4 col-md-offset-8"></div>

2）button的样式去文档里面找，然后如果想在按钮的文字的前面加上小图标（比如：删除有个垃圾箱的图标啦），就去Glyphicons 字体图标里面找

（即在字体前面加一个span就好了）

3）table的样式去文档里面找

<%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %>  
<%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %>  
<**html**>  
<**head**>  
 <**title**>员工列表</**title**>  
 *<%--以/开始的相对路径，找资源，以服务器路径为标准--%>* **<%** pageContext.setAttribute(**"APP\_PATH"**,request.getContextPath());  
 **%>** *<!--引入bootstrap的css样式-->* <**link href="${**APP\_PATH**}/tools/bootstrap-3.3.7-dist/css/bootstrap.min.css" rel="stylesheet"**>  
  
 *<!--引入bootstrap的js文件-->* <**script src="${**APP\_PATH**}/tools/bootstrap-3.3.7-dist/js/bootstrap.min.js"**></**script**>  
  
 *<!--引入jquery-->* <**script type="text/javascript" src="${**APP\_PATH**}/tools/jquery/jquery-3.3.1.js"**></**script**>  
</**head**>  
<**body**>  
 *<!--搭建显示页面-->* <**div class="container"**>  
 *<!--标题-->* <**div class="row"**>  
 <**div class="col-md-12"**>  
 <**h1**>SSM-CURD</**h1**>  
 </**div**>  
 </**div**>  
 *<!--添加、删除大按钮-->* <**div class="row"**>  
 <**div class="col-md-4 col-md-offset-8"**>  
 <**button class="btn btn-primary"**>添加</**button**>  
 <**button class="btn btn-danger"**>删除</**button**>  
 </**div**>  
 </**div**>  
 *<!--显示表格信息-->* <**div class="row"**>  
 <**div class="col-md-12"**>  
 <**table class="table table-hover"**>  
 <**tr**>  
 <**th**>#</**th**>  
 <**th**>empName</**th**>  
 <**th**>gender</**th**>  
 <**th**>email</**th**>  
 <**th**>deptName</**th**>  
 <**th**>操作</**th**>  
 </**tr**>  
 <**c:forEach items="${**pageInfo.list**}" var="emp"**>  
 <**tr**>  
 <**td**>**${**emp.empId**}**</**td**>  
 <**td**>**${**emp.empName**}**</**td**>  
 <**td**>**${**emp.gender=**"M"**?**"男"**:**"女"}**</**td**>  
 <**td**>**${**emp.email**}**</**td**>  
 <**td**>**${**emp.department.deptName**}**</**td**>  
 <**td**>  
 <**button class="btn btn-primary btn-sm"**>  
 <**span class="glyphicon glyphicon-pencil" aria-hidden="true"**></**span**>  
 编辑  
 </**button**>  
 <**button class="btn btn-danger btn-sm"**>  
 <**span class="glyphicon glyphicon-trash" aria-hidden="true"**></**span**>  
 删除  
 </**button**>  
 </**td**>  
 </**tr**>  
 </**c:forEach**>  
 </**table**>  
 </**div**>  
  
 </**div**>  
 *<!--显示分页信息-->* <**div class="row"**>  
 *<!--记录条数-->* <**div class="col-md-6"**>  
 当前第**${**pageInfo.pageNum**}**页，总**${**pageInfo.pages**}**,总**${**pageInfo.total**}**条记录  
 </**div**>  
 *<!--分页条信息-->* <**div class="col-md-6"**>  
 <**nav aria-label="Page navigation"**>  
 <**ul class="pagination"**>  
 <**li**><**a href="#"**>首页</**a**></**li**>  
 <**li**>  
 <**a href="#" aria-label="Previous"**>  
 <**span aria-hidden="true"**>**&laquo;**</**span**>  
 </**a**>  
 </**li**>  
 <**c:forEach items="${**pageInfo.navigatepageNums**}" var="pageNum"**>  
 <**li**><**a href="#"**>**${**pageNum**}**</**a**></**li**>  
 </**c:forEach**>  
 <**li**>  
 <**a href="#" aria-label="Next"**>  
 <**span aria-hidden="true"**>**&raquo;**</**span**>  
 </**a**>  
 </**li**>  
 <**li**><**a href="#"**>末页</**a**></**li**>  
 </**ul**>  
 </**nav**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
  
</**body**>  
</**html**>

14.用el表达式写页面内容（其实上面那个写了部分的，比如数据表中各种数据，以及多少的页码），这里主要写当前页码，下面页码处总是将当前页面放在中间，且高亮显示，而且总共显示相邻4页内容，且，第一页的时候没有首页选项，最后一个页面的时候没有末页选项

1)当前页面的页码高亮显示

<**c:forEach items="${**pageInfo.navigatepageNums**}" var="pageNum"**>  
 <**c:if test="${**pageNum==pageInfo.pageNum**}"**>  
 <**li class="active"**><**a href="#"**>**${**pageNum**}**</**a**></**li**>  
 </**c:if**>  
 <**c:if test="${**pageNum!=pageInfo.pageNum**}"**>  
 <**li**><**a href="#"**>**${**pageNum**}**</**a**></**li**>  
 </**c:if**>  
</**c:forEach**>

2）点击页码跳转到对应页面

15.以上各种逻辑都写的很好，但是此处是在browser-server这种架构模式上适用，而如果在Android和ios浏览器上确不能如此处理请求，而且页面的部分每次发起请求需要改动的只有数据表以及分页信息，所以说只需要发送一个ajax请求，并且最终向页面上发送的是json数据，且解析这些json数据成功显示出数据表

下面的逻辑是：修改index.jsp不让他直接发送/emps请求，转发到list.jsp页面，而是在这个页面上发送ajax请求。此时需要处理的东西为

1）新建controller返回值自定义

1当点击添加按钮时，向数据库中发送ajax请求，将数据库中部门的信息搞出来

2.利用前端技术把数据格式的校验完成(正则表达式)

3.当输入姓名之后，发送ajax请求，去验证此用户名是否已经存在，如果已经存在就不让他添加

4.后端校验，jsr303校验

1）首先，找到依赖hibernate validator（注意：jsr303数据校验支持tomcat7及其以上的服务器）