

**Mybatis、Springmvc练习**

**CRM系统**

# 开发环境

数据库：mysql5.5以上版本。

Jdk：1.7

开发环境：Eclipse mars2

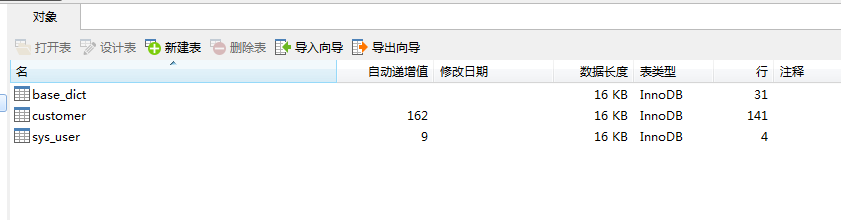
Spring：4.2.4

Mybatis：3.2.7

Tomcat：7

# 数据库

数据库使用mysql 数据库。



1. 创建crm数据库
2. 将参考资料中的sql脚本导入到数据库中

# 工程搭建

工程使用Springmvc、spring、mybatis框架整合完成。

Dao层：SqlMapConfig.xml(空)

applicationContext-dao.xml:数据库连接池、SqlSessionFactory、Mapper的扫描器。

Service层：

配置包扫描器，扫描所有带@Service注解的类。事务管理器、切面。

表现层：

Springmvc.xml:包扫描器@Controller、配置注解驱动、视图解析器。

Jsp：bootstrap

Web.xml:配置spring监听器，前端控制器。

## SqlMapConfig.xml

|  |
| --- |
| <?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>  </configuration> |

## applicationContext-dao.xml

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:task=*"http://www.springframework.org/schema/task"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.2.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-4.2.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-4.2.xsd*  *http://www.springframework.org/schema/task*  *http://www.springframework.org/schema/task/spring-task-4.2.xsd"*>  <!-- 配置 读取properties文件 jdbc.properties -->  <context:property-placeholder location=*"classpath:jdbc.properties"* />  <!-- 配置 数据源 -->  <bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>  <!-- 驱动 -->  <property name=*"driverClassName"* value=*"${jdbc.driver}"* />  <!-- url -->  <property name=*"url"* value=*"${jdbc.url}"* />  <!-- 用户名 -->  <property name=*"username"* value=*"${jdbc.username}"* />  <!-- 密码 -->  <property name=*"password"* value=*"${jdbc.password}"* />  </bean>  <!-- 配置 Mybatis的工厂 -->  <bean class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <!-- 数据源 -->  <property name=*"dataSource"* ref=*"dataSource"* />  <!-- 配置Mybatis的核心 配置文件所在位置 -->  <property name=*"configLocation"* value=*"classpath:SqlMapConfig.xml"* />  <!-- 配置pojo别名 -->  <property name=*"typeAliasesPackage"* value=*"cn.itcast.core.bean"*></property>  </bean>  <!-- 配置 1：原始Dao开发 接口实现类 Mapper.xml 三个 2：接口开发 接口 不写实现类 Mapper.xml 二个 （UserDao、ProductDao  、BrandDao。。。。。。。） 3：接口开发、并支持扫描 cn.itcast.core.dao(UserDao。。。。。） 写在此包下即可被扫描到 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"cn.itcast.core.dao"* />  </bean>  </beans> |

Jdbc.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/crm?characterEncoding=utf-8  jdbc.username=root  jdbc.password=root |

## applicationContext-service.xml

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:task=*"http://www.springframework.org/schema/task"*  xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.2.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-4.2.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-4.2.xsd*  *http://www.springframework.org/schema/task*  *http://www.springframework.org/schema/task/spring-task-4.2.xsd*  *http://code.alibabatech.com/schema/dubbo*  *http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>      <!-- 配置 扫描 @Service -->  <context:component-scan base-package=*"cn.itcast.core.service"*/>        </beans> |

## applicationContext-trans.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.2.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.2.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.2.xsd"*>  <!-- 事务管理器 -->  <bean id=*"transactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 数据源 -->  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!-- 通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 传播行为 -->  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"create\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 切面 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* cn.itcast.core.service.\*.\*(..))"* />  </aop:config>  </beans> |

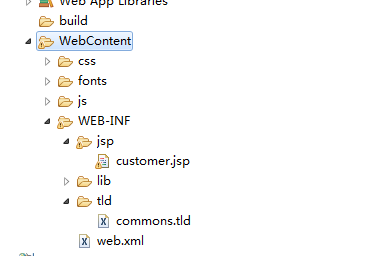
## Springmvc.xml

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:task=*"http://www.springframework.org/schema/task"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.2.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-4.2.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-4.2.xsd*  *http://www.springframework.org/schema/task*  *http://www.springframework.org/schema/task/spring-task-4.2.xsd"*>  <!-- 加载属性文件 -->  <context:property-placeholder location=*"classpath:resource.properties"*/>  <!-- 配置扫描 器 -->  <context:component-scan base-package=*"cn.itcast.core.web.controller"*/>  <!-- 配置处理器映射器 适配器 -->  <mvc:annotation-driven/>    <!-- 配置视图解释器 jsp -->  <bean id=*"jspViewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/jsp/"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean>    </beans> |

## Web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app version=*"2.5"* xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*  *http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>  <welcome-file-list>  <welcome-file>customer.action</welcome-file>  </welcome-file-list>  <!-- 上下文的位置 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext-\*.xml</param-value>  </context-param>  <!-- Spring的监听器 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- POST提交过滤器 UTF-8 -->  <filter>  <filter-name>encoding</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>  </filter>  <filter-mapping>  <filter-name>encoding</filter-name>  <url-pattern>\*.action</url-pattern>  </filter-mapping>  <!-- 前端控制器 -->  <servlet>  <servlet-name>crm</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <!-- 此处不配置 默认找 /WEB-INF/[servlet-name]-servlet.xml -->  <param-value>classpath:springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>crm</servlet-name>  <!-- 1:\*.do \*.action 拦截以.do结尾的请求 (不拦截 jsp png jpg .js .css) 2:/ 拦截所有请求  （不拦截.jsp) 建议使用此种 方式 （拦截 .js.css .png) (放行静态资源） 3:/\* 拦截所有请求（包括.jsp) 此种方式 不建议使用 -->  <url-pattern>\*.action</url-pattern>  </servlet-mapping>  </web-app> |

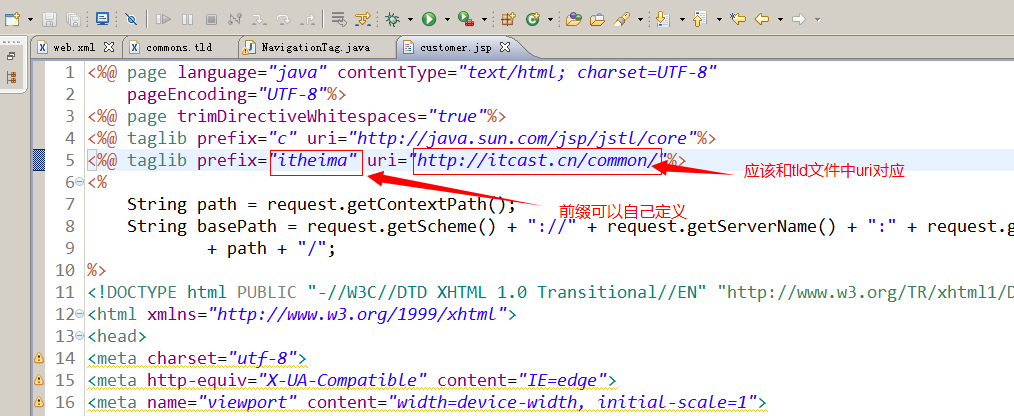
## 加入jsp及分页标签

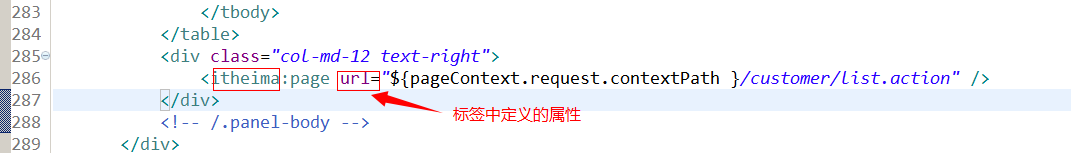


Tld文件需要放到WEB-INF目录下， tomcat的规定。当tomcat启动时会自动加载。



Jsp中使用标签：





# 查询条件初始化

## 需求



初始化查询条件下拉列表。

## Sql

SELECT \* from base\_dict WHERE dict\_type\_code='006'

## Dao

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace=*"com.itheima.crm.dao.BaseDictDao"*>  <select id=*"getBaseDictList"* parameterType=*"string"* resultType=*"BaseDict"*>  SELECT \* from base\_dict WHERE dict\_type\_code=#{typeCode}  </select>  </mapper> |

## Service

|  |
| --- |
| @Service  **public** **class** BaseDictServiceImpl **implements** BaseDictService {  @Autowired  **private** BaseDictDao baseDictDao;    @Override  **public** List<BaseDict> getBaseDictList(String typeCode) {  List<BaseDict> list = baseDictDao.getBaseDictList(typeCode);  **return** list;  }  } |

## Controller

Spring容器由ContextLoaderListener  
父容器

Dao

server

子容器

Springmvc

前端控制器初始化而来。

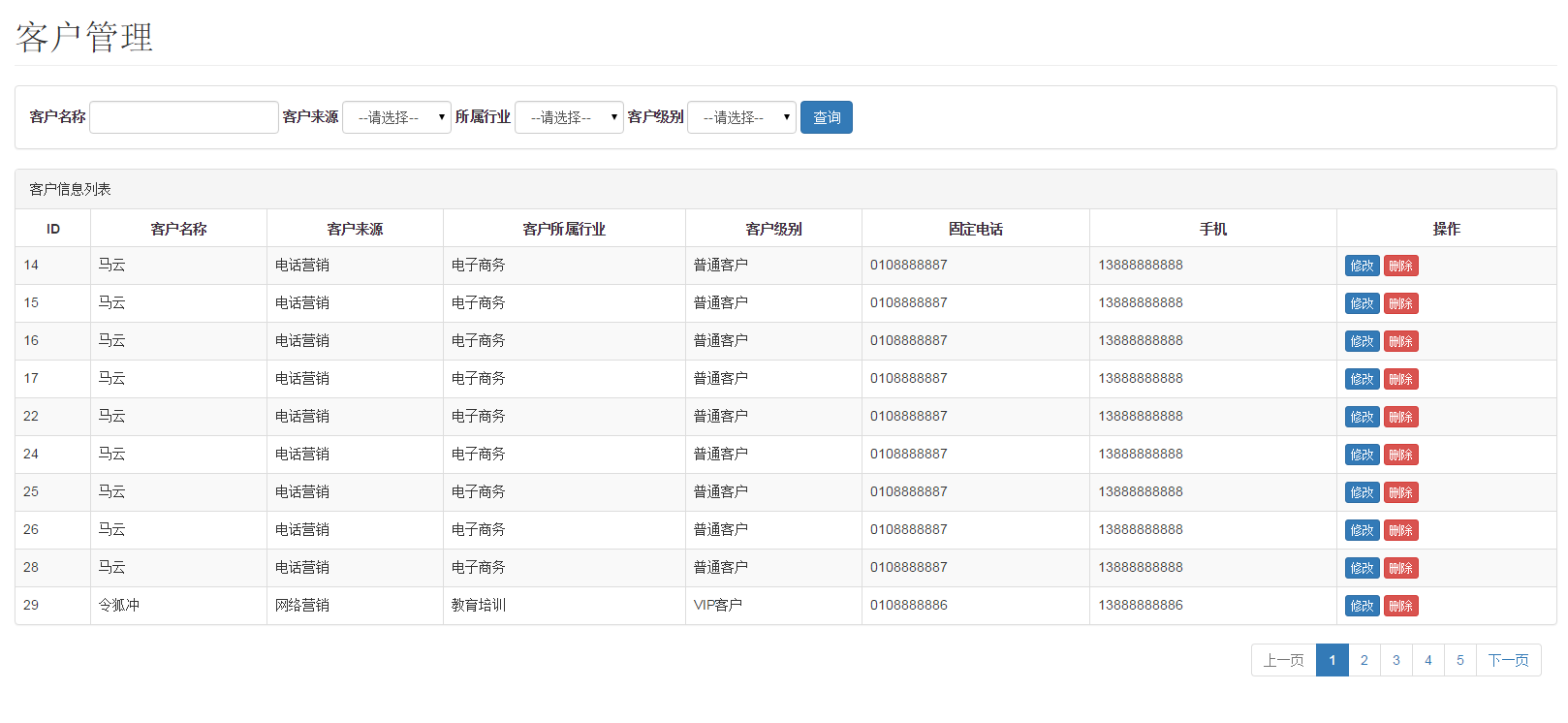
Controller

规则：子容器可以访问父容器的对象，父容器不能访问子容器的对象。

|  |
| --- |
| **public** **class** CustomerController {    @Autowired  **private** BaseDictService baseDictService;  @Value("${customer.source.code}")  **private** String CustomerSourceCode;  @Value("${customer.industry.code}")  **private** String CustomerIndustryCode;  @Value("${customer.level.code}")  **private** String CustomerLevelCode;  @RequestMapping("/list")  **public** String showCustomerList(Model model) {  //查询字典表初始化下拉列表  List<BaseDict> custSourceList = baseDictService.getBaseDictList(CustomerSourceCode);  List<BaseDict> custIndustryList = baseDictService.getBaseDictList(CustomerIndustryCode);  List<BaseDict> custLevelList = baseDictService.getBaseDictList(CustomerLevelCode);  //把列表传递给jsp页面  model.addAttribute("fromType", custSourceList);  model.addAttribute("industryType", custIndustryList);  model.addAttribute("levelType", custLevelList);    **return** "customer";  }  } |

# 客户列表展示

## 需求



展示商品列表，并且可以根据查询条件过滤查询结果，并且实现分页处理。

## Sql

|  |
| --- |
| SELECT  a.cust\_id,  a.cust\_name,  a.cust\_user\_id,  a.cust\_create\_id,  b.dict\_item\_name cust\_source,  c.dict\_item\_name cust\_industry,  d.dict\_item\_name cust\_level,  a.cust\_linkman,  a.cust\_phone,  a.cust\_mobile,  a.cust\_zipcode,  a.cust\_address,  a.cust\_createtime  FROM  customer a  LEFT JOIN base\_dict b ON a.cust\_source = b.dict\_id  LEFT JOIN base\_dict c ON a.cust\_industry = c.dict\_id  LEFT JOIN base\_dict d ON a.cust\_level = d.dict\_id  WHERE  cust\_name LIKE '%马%'  AND cust\_source = 6  AND cust\_industry = 2  AND cust\_level = 22 |

## Dao

|  |
| --- |
| <select id=*"getCustList"* parameterType=*"QueryVo"* resultType=*"customer"*>  SELECT  a.cust\_id,  a.cust\_name,  a.cust\_user\_id,  a.cust\_create\_id,  b.dict\_item\_name cust\_source,  c.dict\_item\_name cust\_industry,  d.dict\_item\_name cust\_level,  a.cust\_linkman,  a.cust\_phone,  a.cust\_mobile,  a.cust\_zipcode,  a.cust\_address,  a.cust\_createtime  FROM  customer a  LEFT JOIN base\_dict b ON a.cust\_source = b.dict\_id  LEFT JOIN base\_dict c ON a.cust\_industry = c.dict\_id  LEFT JOIN base\_dict d ON a.cust\_level = d.dict\_id  <where>  <if test=*"custName!=null and custName!=''"*>  AND cust\_name LIKE '%${custName}%'  </if>  <if test=*"custSource!=null and custSource!=''"*>  AND cust\_source = #{custSource}  </if>  <if test=*"custIndustry!=null and custIndustry!=''"*>  AND cust\_industry = #{custIndustry}  </if>  <if test=*"custLevel!=null and custLevel!=''"*>  AND cust\_level = #{custLevel}  </if>  </where>  LIMIT #{start},#{rows}  </select> |

增加count后的dao

|  |
| --- |
| <mapper namespace=*"com.itheima.crm.dao.CustomerDao"*>    <sql id=*"cust\_query\_where"*>  <where>  <if test=*"custName!=null and custName!=''"*>  AND cust\_name LIKE '%${custName}%'  </if>  <if test=*"custSource!=null and custSource!=''"*>  AND cust\_source = #{custSource}  </if>  <if test=*"custIndustry!=null and custIndustry!=''"*>  AND cust\_industry = #{custIndustry}  </if>  <if test=*"custLevel!=null and custLevel!=''"*>  AND cust\_level = #{custLevel}  </if>  </where>  </sql>    <select id=*"getCustList"* parameterType=*"QueryVo"* resultType=*"customer"*>  SELECT  a.cust\_id,  a.cust\_name,  a.cust\_user\_id,  a.cust\_create\_id,  b.dict\_item\_name cust\_source,  c.dict\_item\_name cust\_industry,  d.dict\_item\_name cust\_level,  a.cust\_linkman,  a.cust\_phone,  a.cust\_mobile,  a.cust\_zipcode,  a.cust\_address,  a.cust\_createtime  FROM  customer a  LEFT JOIN base\_dict b ON a.cust\_source = b.dict\_id  LEFT JOIN base\_dict c ON a.cust\_industry = c.dict\_id  LEFT JOIN base\_dict d ON a.cust\_level = d.dict\_id  <include refid=*"cust\_query\_where"*/>  LIMIT #{start},#{rows}  </select>  <select id=*"getCustListCount"* parameterType=*"QueryVo"* resultType=*"int"*>  SELECT count(\*)  FROM  customer a  LEFT JOIN base\_dict b ON a.cust\_source = b.dict\_id  LEFT JOIN base\_dict c ON a.cust\_industry = c.dict\_id  LEFT JOIN base\_dict d ON a.cust\_level = d.dict\_id  <include refid=*"cust\_query\_where"*/>  </select>  </mapper> |

## Service

根据查询条件查询数据库得到客户列表。分页条件。

接收查询条件QueryVo接收，使用page接收页码。

1. 通过page计算start。
2. 调用dao查询客户列表。
3. 做count处理。计算出此查询条件中共查询到多少条记录。

返回结果：Page对象。

条件：QueryVo

|  |
| --- |
| @Service  **public** **class** CustomerServiceImpl **implements** CustomerService {  @Autowired  **private** CustomerDao customerDao;    @Override  **public** Page<Customer> getCustList(QueryVo queryVo) {  queryVo.setStart((queryVo.getPage() - 1) \* queryVo.getRows());  List<Customer> custList = customerDao.getCustList(queryVo);  Page<Customer> page = **new** Page<Customer>();  //设置客户列表  page.setRows(custList);  page.setPage(queryVo.getPage());  page.setSize(queryVo.getRows());  //计算查询总记录数  **int** total = customerDao.getCustListCount(queryVo);  page.setTotal(total);  **return** page;  }  } |

## Controller

### 分析

1、接收页面提交的查询参数：

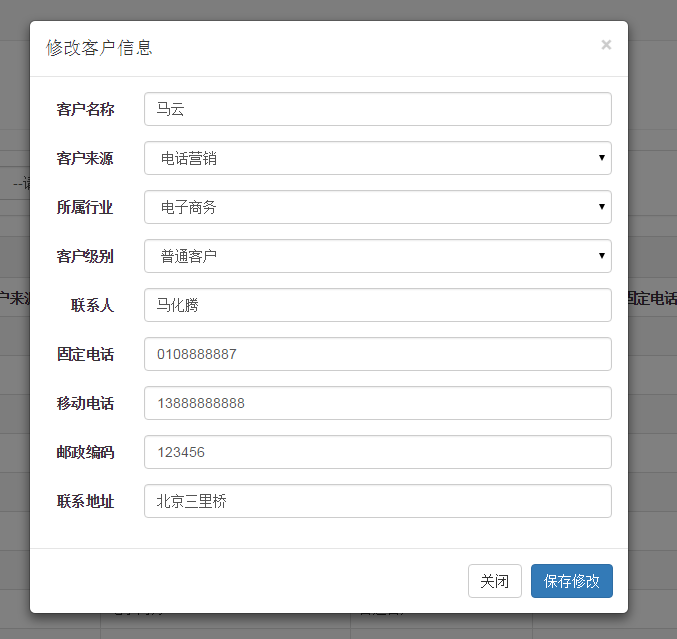
保证jsp页面提交的表单中的input 的name属性和QueryVo中的属性一致

1. 调用Service查询客户列表
2. 把客户列表传递给页面。

|  |
| --- |
| @RequestMapping("/list")  **public** String showCustomerList(Model model, QueryVo queryVo) **throws** Exception {    String custName = queryVo.getCustName();  **if** (custName != **null** && !"".equals(custName)) {  custName = **new** String(custName.getBytes("iso8859-1"), "utf-8");  queryVo.setCustName(custName);  }  //查询字典表初始化下拉列表  List<BaseDict> custSourceList = baseDictService.getBaseDictList(CustomerSourceCode);  List<BaseDict> custIndustryList = baseDictService.getBaseDictList(CustomerIndustryCode);  List<BaseDict> custLevelList = baseDictService.getBaseDictList(CustomerLevelCode);  //查询客户列表  Page<Customer> page = customerService.getCustList(queryVo);  //把page放到request中  model.addAttribute("page", page);    //把列表传递给jsp页面  model.addAttribute("fromType", custSourceList);  model.addAttribute("industryType", custIndustryList);  model.addAttribute("levelType", custLevelList);    //页面回显  model.addAttribute("custName", queryVo.getCustName());  model.addAttribute("custSource", queryVo.getCustSource());  model.addAttribute("custIndustry", queryVo.getCustIndustry());  model.addAttribute("custLevel", queryVo.getCustLevel());      **return** "customer";  } |

# 修改客户信息

## 需求



1. 点击客户列表中的“修改”按钮弹出客户信息修改对话框，并初始化客户信息
2. 点击“保存修改”按钮将修改后的结果保存到数据库中

## 展示客户信息

### 分析

请求的url：

customer/edit.action

参数：cust\_id客户id

返回值：响应json数据，直接由Customer转换而来。需要使用@ResponseBody注解。

### Dao层

根据客户id查询客户信息。

|  |
| --- |
| <select id=*"getCustomerById"* parameterType=*"long"* resultType=*"customer"*>  select \* from customer where cust\_id = #{custId}  </select> |

### Service层

|  |
| --- |
| @Override  **public** Customer getCustomerById(**long** custId) {  Customer customer = customerDao.getCustomerById(custId);  **return** customer;  } |

### Controller

要求返回json数据

请求的url：

customer/edit.action

参数：cust\_id客户id

返回值：响应json数据

|  |
| --- |
| @RequestMapping("/edit")  @ResponseBody  **public** Customer getCustomerById(Long id) {  Customer customer = customerService.getCustomerById(id);  **return** customer;  } |

## 提交修改

### 分析

请求的url：customer/update.action

请求的方法：post

参数：表单中的数据。

返回结果：OK

### Dao层

|  |
| --- |
| <update id=*"updateCustomerById"* parameterType=*"customer"*>  UPDATE customer  <set>  <if test=*"cust\_name!=null"*>  cust\_name=#{cust\_name},  </if>  <if test=*"cust\_user\_id!=null"*>  cust\_user\_id=#{cust\_user\_id},  </if>  <if test=*"cust\_create\_id!=null"*>  cust\_create\_id=#{cust\_create\_id},  </if>  <if test=*"cust\_source!=null"*>  cust\_source=#{cust\_source},  </if>  <if test=*"cust\_industry!=null"*>  cust\_industry=#{cust\_industry},  </if>  <if test=*"cust\_level!=null"*>  cust\_level=#{cust\_level},  </if>  <if test=*"cust\_linkman!=null"*>  cust\_linkman=#{cust\_linkman},  </if>  <if test=*"cust\_phone!=null"*>  cust\_phone=#{cust\_phone},  </if>  <if test=*"cust\_mobile!=null"*>  cust\_mobile=#{cust\_mobile},  </if>  <if test=*"cust\_zipcode!=null"*>  cust\_zipcode=#{cust\_zipcode},  </if>  <if test=*"cust\_address!=null"*>  cust\_address=#{cust\_address},  </if>  <if test=*"cust\_createtime!=null"*>  cust\_createtime=#{cust\_createtime},  </if>  </set>  WHERE  cust\_id = #{cust\_id}  </update> |

### Service层

|  |
| --- |
| @Override  **public** **void** updateCustomerById(Customer customer) {  customerDao.updateCustomerById(customer);  } |

### Controller

接收参数：Customer接收。

响应结果：OK字符串。使用@ResponseBody

请求的url：customer/update.action

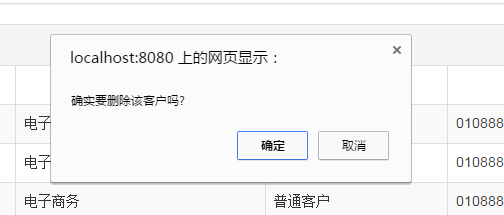
|  |
| --- |
| @RequestMapping(value="/update", method=RequestMethod.***POST***)  @ResponseBody  **public** String updateCustomer(Customer customer) {  customerService.updateCustomerById(customer);  //直接向浏览器响应字符串需要使用@ResponseBody  **return** "OK";    } |

# 删除客户

## 需求



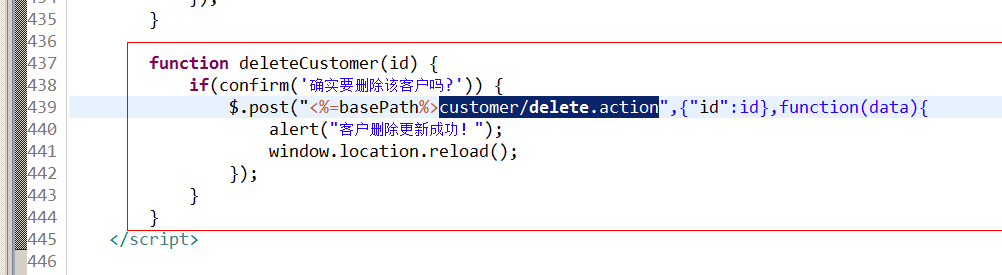
点击客户列表中的删除按钮，提示“警告信息”



点击确定后删除用户信息，并刷新页面。

分析：

请求的url：customer/delete.action



参数：id（客户id）

响应的内容：OK（字符串需要使用@ResponseBody）