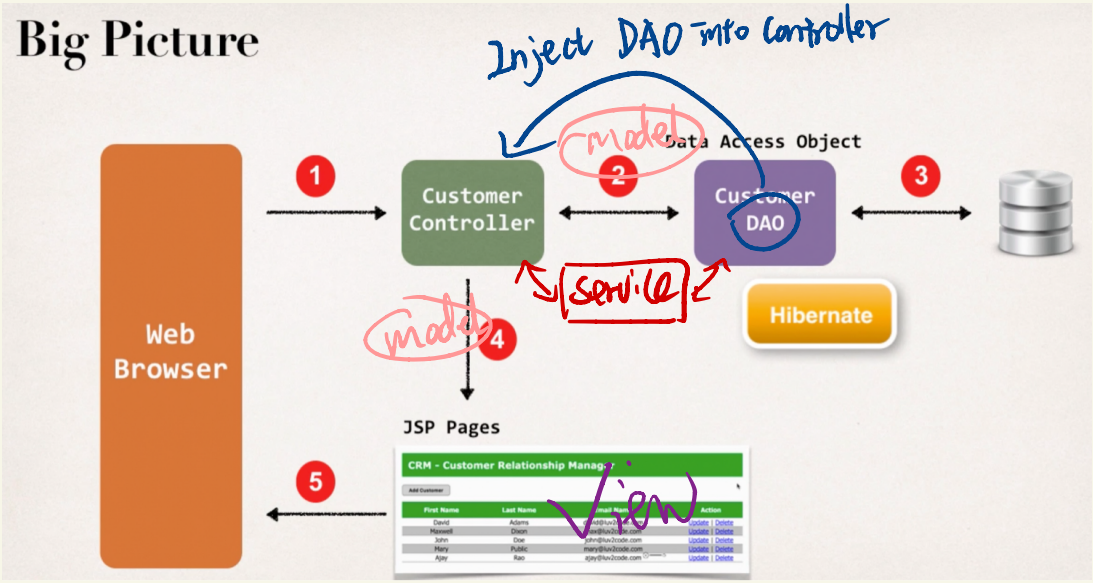
CRM project

Steps:

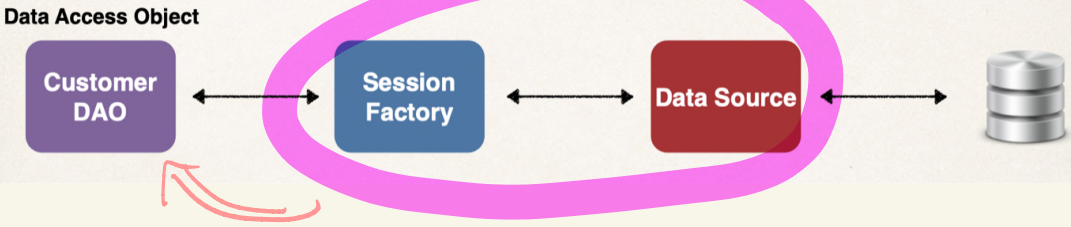
1. set up database
2. Test DB connecti...
3. Setup Developme...
4. List customers
5. view
6. Define Services w...
7. Add Customer
8. Update Customer
9. Delete Customer



- Our Hibernate Session Factory needs a Data Source
  - The data source defines database connection info

IoC container

Beans



IoC / DI

Spring:

Model 自动创建，存储键值对 用来传输数据.

eg. model.addAttribute("customer", customers)

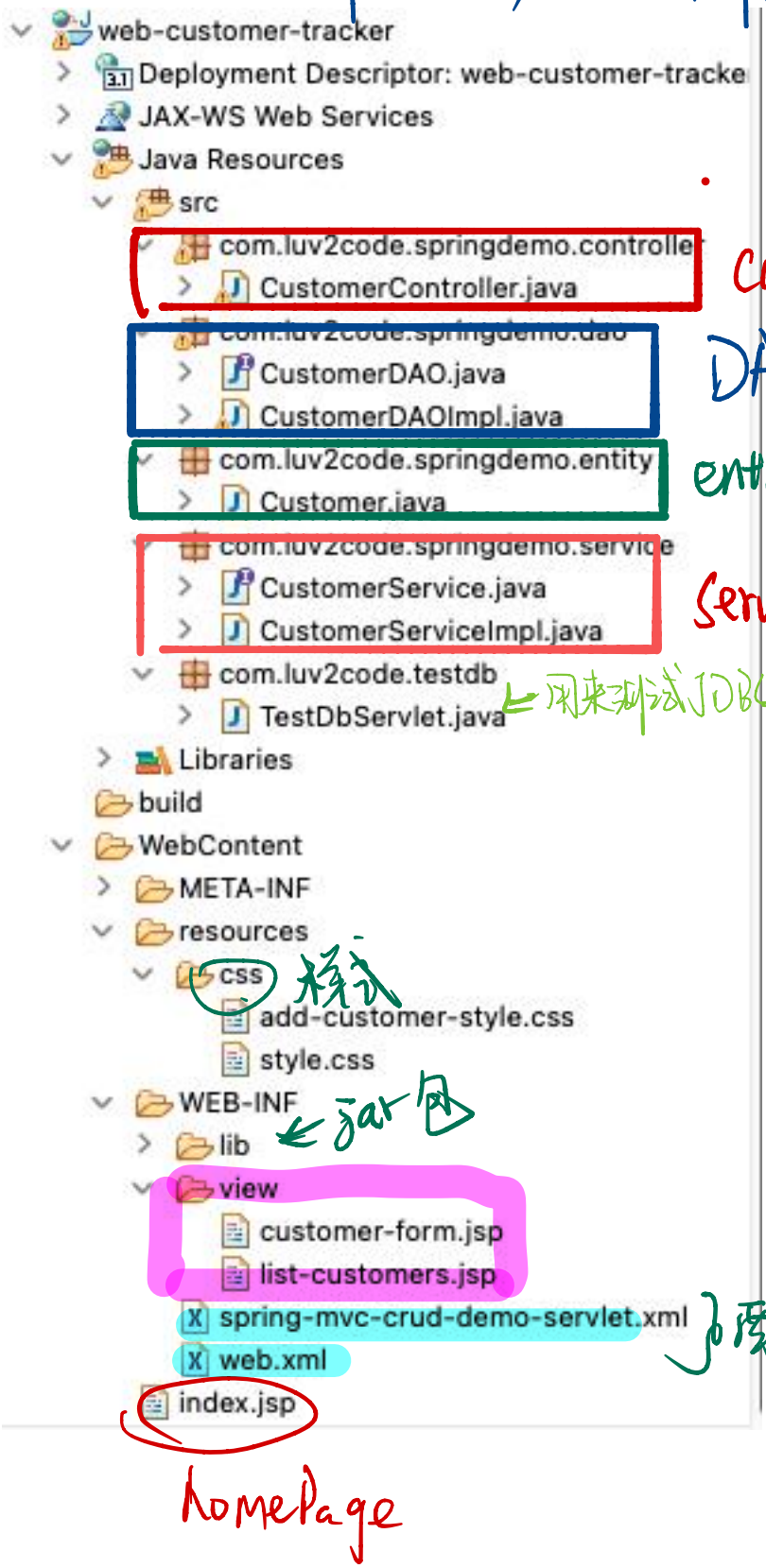
name value

controller：将不同 model 显示在不同的 view 上

Controller ⟷ Service ⟷ DAO ⟷ DB

(interface implement) (interface implementation) (entity)

Project structure:



Configuration files:

1.spring configuration file

spring-mvc-crud-demo-servlet

CustomerControll | spring-mvc-crud | list-customers.j | CustomerDAO.jav | CustomerService. | CustomerServiceI | customer-form.j

```xml
<?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:tx="http://www.springframework.org/schema/tx"
    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
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- Add support for component scanning -->
    <context:component-scan base-package="com.luv2code.springdemo" />

    <!-- Add support for conversion, formatting and validation support -->
    <mvc:annotation-driven/>

    <!-- Define Spring MVC view resolver -->
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/view/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- Step 1: Define Database DataSource / connection pool -->
    <bean id="myDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
        destroy-method="close">
        <property name="driverClass" value="com.mysql.cj.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/web_customer_tracker?useSSL=false&amp;serverTimezone=UTC" />
        <property name="user" value="springstudent" />
        <property name="password" value="springstudent" />

        <!-- these are connection pool properties for C3P0 -->
        <property name="minPoolSize" value="5" />
        <property name="maxPoolSize" value="20" />
        <property name="maxIdleTime" value="30000" />
    </bean>

    <!-- Step 2: Setup Hibernate session factory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
        <property name="dataSource" ref="myDataSource" />
        <property name="packagesToScan" value="com.luv2code.springdemo.entity" />
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
            </props>
        </property>
    </bean>

    <!-- Step 3: Setup Hibernate transaction manager -->
    <bean id="myTransactionManager"
        class="org.springframework.orm.hibernate5.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <!-- Step 4: Enable configuration of transactional behavior based on annotations -->
    <tx:annotation-driven transaction-manager="myTransactionManager" />

    <!-- add support for reading web resources: css, images, js, etc... -->
    <mvc:resources location="/resources/" mapping="/resources/**"></mvc:resources>

</beans>
```

直接用模板

Scan bean

解析 view

常略前后缀

DB ↔ DataSource ↔ sessionFactory ↔ DAO

mySql 8.0

常用配置

如果scan多个package，用逗号隔开

将会扫描所有 annotated @Entity classes

spring automatically begin and end a transaction.

** 包含 subdirectories（递归写法）.

name = variable

value = value

2.web.xml

CustomerControl | spring-mvc-crud | list-customers. | CustomerDAO.jav | CustomerService | CustomerService

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
id="WebApp_ID" version="3.1">
  <display-name>spring-mvc-crud-demo</display-name>

  <absolute-ordering />

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring-mvc-crud-demo-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

welcome page

从上到下 scan

spring 的配置文件

处理 browser 发来的 request

处理所有请求

Spring MVC 替代 Servlet 处理请求、响应请求，获取表单参数，表单校验等，

使用步骤：

① 配置 DispatcherServlet ~> 相当于总转发器，用于接收响应请求

② 配置 HandlerMapping ~> 根据 url 查找 handler (controller)

③ 配置 HandlerAdapter ~> 执行 controller

④ 配置 ViewResolver ~> 解析成 view

⑤ 编写 view ~> html, jsp

⑥ 编写 controller、service、Dao

自己开发

Create hibernate entity (Object - to Relational Mapping ORM)

Java :

Customer.java mapped to a database table

```java
package com.luv2code.springdemo.entity;

import javax.persistence.Column;

@Entity
@Table(name="customer")
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="first_name")
    private String firstName;

    @Column(name="last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    public Customer() {

    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Customer [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", email=" + email + "]";
    }


}
```

table name

fields

DB ↔ data source ↔ session factory

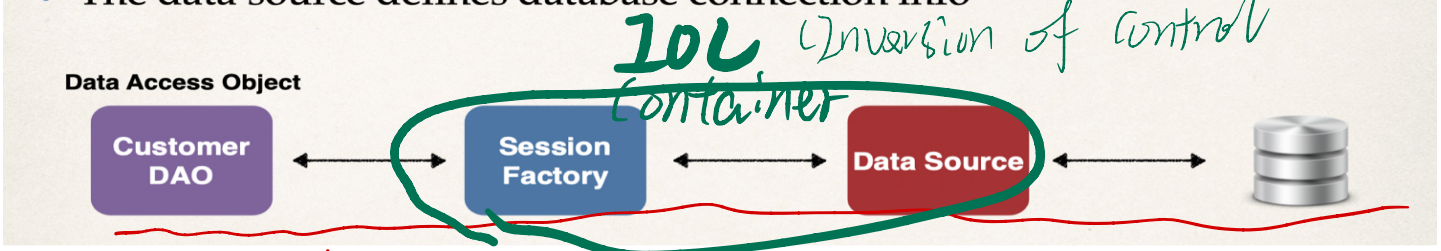
Data access object

CustomerDAO.jav | CustomerControll | spring-mvc-crud

```java
package com.luv2code.springdemo.dao;

import java.util.List;

public interface CustomerDAO {

    public List<Customer> getCustomers();

    public void saveCustomer(Customer customer);

    public Customer getCustomer(int theId);

    public void deleteCustomer(int theId);
}
```

CRUD = Create, Read, update, delete

- Our Hibernate Session Factory needs a Data Source
- The data source defines database connection info



all dependencies

⇓

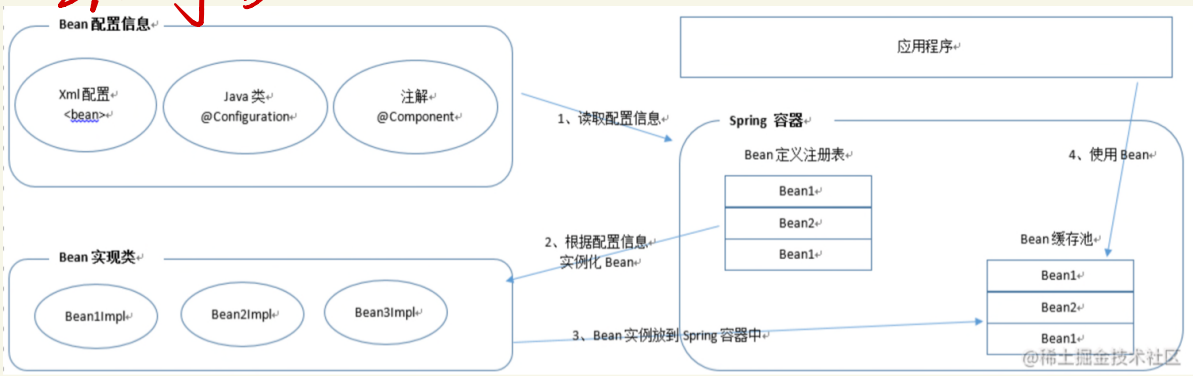
wire together with Dependency Injection (DI)

@component (表面组件)

@controller @Service @Repository ⇒ 即 bean

Bean 是由 IoC 容器初始化、装配及管理的对象，是组件类和实体类 (POJO) 对象的总称

Spring Bean



使用@Autowired实现自动装配

注解的位置：

1) class fields
2) set 方法
3) constructor

实现类.

CustomerDAOImpl.java | CustomerController.j | spring-mvc-crud-de | list-customers.jsp | CustomerService.java | CustomerServiceImpl

```java
import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.query.Query;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.luv2code.springdemo.entity.Customer;

@Repository
public class CustomerDAOImpl implements CustomerDAO {


    //need to inject the session factory
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    //@Transactional move this functionality to service layer to handle
    public List<Customer> getCustomers() {
        // get the current hibernate session

        Session currentSession = sessionFactory.getCurrentSession();
        //create a query,sort by last name

        Query<Customer> theQuery = currentSession.createQuery("from Customer order by lastName",Customer.class);
        //execute query and get result list

        List<Customer> customers = theQuery.getResultList();
        //return result
        return customers;
    }

    @Override
    public void saveCustomer(Customer customer) {

        //get current hibernate session
        Session session= sessionFactory.getCurrentSession();
        //save or update the customer
        session.saveOrUpdate(customer);

    }

    @Override
    public Customer getCustomer(int theId) {
        // get the current hibernate session
        Session session= sessionFactory.getCurrentSession();

        //now retrieve /read from database using the primary key
        Customer customer = session.get(Customer.class, theId);
        return customer;
    }

    @Override
    public void deleteCustomer(int theId) {
        // get the current hibernate session
        Session session= sessionFactory.getCurrentSession();


        //delete the object with primary key
        Query theQuery = session.createQuery("delete from Customer where id=:customerId");

        theQuery.setParameter("customerId", theId);
        theQuery.executeUpdate();
    }

}
```

省略 session begin 和 close

类名

一致

Service layer = delegate calls to DAO

CustomerController.java | spring-mvc-crud-demo-servlet.xml | list-customers.jsp | CustomerService.java

```java
package com.luv2code.springdemo.service;

import java.util.List;

public interface CustomerService {

    public List<Customer> getCustomers();

    public void saveCustomer(Customer customer);

    public Customer getCustomer(int theId);

    public void deleteCustomer(int theId);
}
```

CustomerController.java | CustomerServiceImpl.java | spring-mvc-crud-d

```java
package com.luv2code.springdemo.service;

import java.util.List;


@Service
public class CustomerServiceImpl implements CustomerService {


    //inject customer DAO
    @Autowired
    private CustomerDAO customerDAO;



    @Override
    @Transactional
    public List<Customer> getCustomers() {
        return customerDAO.getCustomers();
    }



    @Override
    @Transactional
    public void saveCustomer(Customer customer) {

        customerDAO.saveCustomer(customer);

    }



    @Override
    @Transactional
    public Customer getCustomer(int theId) {

        return customerDAO.getCustomer(theId);
    }



    @Override
    @Transactional
    public void deleteCustomer(int theId) {
         customerDAO.deleteCustomer(theId);
    }

}

```

Project name: web-customer-tracker

CustomerController.java | spring-mvc-crud-demo-servlet.xml | list-customers.jsp | customer-form.jsp | add-custom

```java
package com.luv2code.springdemo.controller;

import java.util.List;

@Controller
@RequestMapping("/customer")
public class CustomerController {

    //remove this part, use service layer instead DAO
//  //need to inject the customer DAO
//  @Autowired
//  private CustomerDAO customerDAO;

    //inject customer service
    @Autowired
    private CustomerService customerService;

    @GetMapping("/list")
    public String listCustomers(Model theModel) {

        //get customers from the service,service delegate calls to DAO
        List<Customer> customers = customerService.getCustomers();
        //add the customers to the model

        theModel.addAttribute("customers",customers);

        return "list-customers";
    }

    @GetMapping("/showFormForAdd")
    public String showFormForAdd(Model theModel) {
        Customer customer = new Customer();
        theModel.addAttribute("customer",customer);
        return "customer-form";
    }

    @PostMapping("/saveCustomer")
    public String saveCustomer(@ModelAttribute("customer") Customer customer) {
        customerService.saveCustomer(customer);
        return "redirect:/customer/list";

    }

    @GetMapping("/showFormForUpdate")
    public String showFormForUpdate(@RequestParam("customerId") int theId, Model theModel) {
        //get the customer form the service
        Customer customer= customerService.getCustomer(theId);
        //set customer as a model attribute to pre-populate the form
        theModel.addAttribute("customer", customer);
        //send over to form
        return "customer-form";
    }

    @GetMapping("/delete")
    public String deleteCustomer(@RequestParam("customerId") int theId) {
        //delete the customer

        customerService.deleteCustomer(theId);

        return "redirect:/customer/list";
    }
}
```

URL路径：web-customer-tracker/customer/list

/showFormForAdd

/saveCustomer

View:

Homepage

index.jsp | spring-mvc-crud-demo-servlet.xml | list-c

```jsp
<% response.sendRedirect("customer/list"); %>
```

转向到

index.jsp | spring-mvc-crud-demo-servlet.xml | list-customers.jsp | *customer-form.jsp | add-customer-style.css | style.css

```jsp
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<!DOCTYPE html>
<html>
    <head>
        <title>Save Customer</title>
        <link type="text/css" rel="stylesheet" href= "${pageContext.request.contextPath }/resources/css/style.css"/>
        <link type="text.css" rel="stylesheet" href="${pageContext.request.contextPath }/resources/css/add-customer-style.css"/>

    </head>
<body>

    <div id = "wrapper">
        <div id="header">
        <h2>CRM - Customer Relationship Manager</h2>
        </div>
    </div>

    <div id= "container">
        <h3>Save Customer</h3>

        <form:form action="saveCustomer" modelAttribute ="customer" method="POST">
        <!-- need to associate this data with customer id -->
            <form:hidden path="id"/>

            <table>
            <tbody>
                <tr>
                    <td> <label>First name:</label></td>
                    <td><form:input path="firstName"/></td>
                </tr>
                <tr>
                    <td> <label>Last name:</label></td>
                    <td><form:input path="lastName"/></td>
                </tr>
                <tr>
                    <td> <label>Email:</label></td>
                    <td><form:input path="email"/></td>
                </tr>
                <tr>
                    <td> <label></label></td>
                    <td><input type ="submit" value= "save" class="save"></td>
                </tr>
            </tbody>

            </table>
        </form:form>

        <div style="clear;both;"></div>

        <p>
            <a href="${pageContext.request.contextPath}/customer/list">Back to List</a>
        </p>

    </div>

</body>
</html>
```

@RequestMapping( ) 根对路径 (/customer/saveCustomer)

action 类是用户请求和业务逻辑之间的桥梁

index.jsp | spring-mvc-crud-demo-servlet.xml | *list-customers.jsp | add-customer-style.css | style.css

```jsp
<!DOCTYPE html>

<html>

<head>
<title>List Customers</title>

<!-- reference our style sheet -->
<link type="text/css" rel="stylesheet" href= "${pageContext.request.contextPath }/resources/css/style.css"/>

</head>
<body>

    <div id = "wrapper">
        <div id="header">
        <h2>CRM - Customer Relationship Manager</h2>
        </div>
    </div>

    <div id="container">
        <div id="container">

        <!-- put new button: Add Customer -->

        <input type="button" value ="Add Customer"
            onClick="window.location.href='showFormForAdd';return false;"
            class="add-button"
        />
        <!--  add out html table here -->
        <table>
        <tr>
        <th> First Name</th>
        <th> Last Name</th>
        <th> Email</th>
        <th> Action</th>

        </tr>
            <!-- loop over and print our customers -->
            <c:forEach var = "tempCustomer" items ="${customers}">

            <!-- construct an "update" link with customer id -->
            <c:url var="updateLink" value="/customer/showFormForUpdate">
                <c:param name="customerId" value="${tempCustomer.id}"/>
            </c:url>

            <!-- construct an "delete" link with customer id -->
            <c:url var="deleteLink" value="/customer/delete">
                <c:param name="customerId" value="${tempCustomer.id}"/>
            </c:url>

            <tr>
                <td>${tempCustomer.firstName } </td>
                <td>${tempCustomer.lastName } </td>
                <td>${tempCustomer.email } </td>
                <td>
                <!-- display the update link -->
                <a href="${updateLink}">Update</a>
                |
                <a href="${deleteLink}" onClick="if(!(confirm('Are you sure to delete this customer?'))) return false">Delete</a>
                </td>

            </tr>
            </c:forEach>

        </table>

        </div>
    </div>
</body>
</html>
```

Create --> get FirstName
get LastName
set Email

submit --> set FirstName
set LastName
set Email