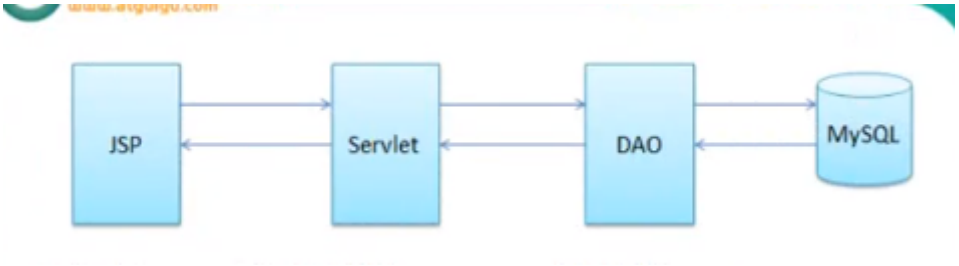
基础架构
编写DAO操作
编写Servlet
一个Servlet处理多个请求(通过反射解决)
模糊查询
删除操作
增加新Customer操作
修改数据

基础架构



1.首先编写数据库操作
2.编写servlet操作
3.最后编写jsp

编写DAO操作

- 加入c3p0数据源
- 编写jdbcUtils（测试是否能够成功连接）
- 编写dao（导入dbutils包来简化具体的dao操作）
- 编写CustomerDao（DAO Data access objects）

```
1
2  //获取<T>具体是哪个类
3  public DAO(){
4   Type superClassType = getClass().getGenericSuperclass();
5
6   if (superClassType instanceof ParameterizedType) {
```

```
 ParameterizedType parameterizedType = (ParameterizedType)
superClassType;

 //获取到T的数组
  Type[] typeArgs = parameterizedType.getActualTypeArguments();
  if (typeArgs != null && typeArgs.length > 0) {
  clazz = (Class<T>) typeArgs[0];
  System.out.println(clazz);
  }
  }
}
```

```
public class CustomerDaoImp extends DAO<Customer> implements CustomerDao
{}
```

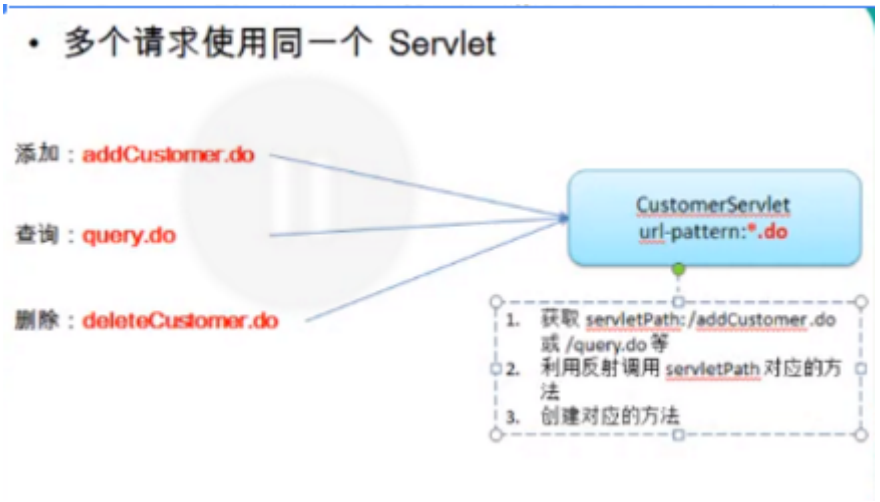
测试结果

```
public class CustomerDaoImpTest {

 @Test
  public void get() {
  CustomerDao customerDao = new CustomerDaoImp();
  }
}
//结果
class bean.Customer
```

编写Servlet

一个Servlet处理多个请求(通过反射解决)



将访问路径设置为*.do

```
<servlet-mapping>
 <servlet-name>CustomerServlet</servlet-name>
 <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

```
<a href="addCustomer.do">Add</a>
<a href="query.do">Query</a>
<a href="deleteCustomer.do">Delete</a>
```

实际访问连接

```
http://localhost:8081/webtest/addCustomer.do
```

所以可以利用反射来调用对应的方法，注意这里反射的参数类型直接使用HttpServletRequest.class,而不要使用req.getClass，因为动态调用的时候，req可以是它的子类。

```
String path = req.getServletPath();
System.out.println(path);
String methodName = path.substring(path.indexOf("/") + 1,
path.indexOf("."));
//通过反射调用对应的方法
Method method = null;
try {
 method = getClass().getDeclaredMethod(methodName, HttpServletRequest.cla
ss, HttpServletResponse.class);
 method.invoke(this, req, resp);
} catch (NoSuchMethodException e) {
 e.printStackTrace();
 } catch (IllegalAccessException e) {
 e.printStackTrace();
 } catch (InvocationTargetException e) {
 e.printStackTrace();
 }
```

使用正则表达式修改获取method的方法

```
Pattern pattern = Pattern.compile("\\/(\\w+).do$");
Matcher matcher = pattern.matcher("/jsp/addCustomer.do");
String methodName = null;
if (matcher.find()) {
 methodName = matcher.group(1);
```

```
}
```

```
private void addCustomer(HttpServletRequest req, HttpServletResponse resp) {
 System.out.println("addCustomer");
}
```

模糊查询

通过 like %% 来进行模糊查询

我们创建一个辅助类，来存储模糊字段

```
//添加模糊查询的方法
private String get(String value) {
if (value == null) {
value = "%%";
} else {
value = "%" + value + "%";
}
return value;
}
```

```
public class CriteriaCustomer {
 private String name;
 private String phone;
 private String address;

 public CriteriaCustomer() {
 }

 public CriteriaCustomer(String name, String phone, String address) {
  this.name = name;
  this.address = address;
  this.phone = phone;
  }

  //添加模糊查询的方法
  private String get(String value) {
  if (value == null) {
  value = "%%";
```

```
19   } else {
20   value = "%" + value + "%";
21   }
22   return value;
23   }
24
25   public String getName() {
26   return get(name);
27   }
28
29   public String getPhone() {
30   return get(phone);
31   }
32
33   public String getAddress() {
34   return get(address);
35   }
36   }
```

通过这个类，我们可以简化数据库查询的方法

```
1  @Override
2  public List<Customer> getWithIndistinct(CriteriaCustomer
criteriaCustomer) {
3   String sql = "select id,name,phone,address from customers where name lik
e ? and phone like ? and address like ?";
4   return getForList(sql, criteriaCustomer.getName(), criteriaCustomer.getP
hone(), criteriaCustomer.getAddress());
5  }
```

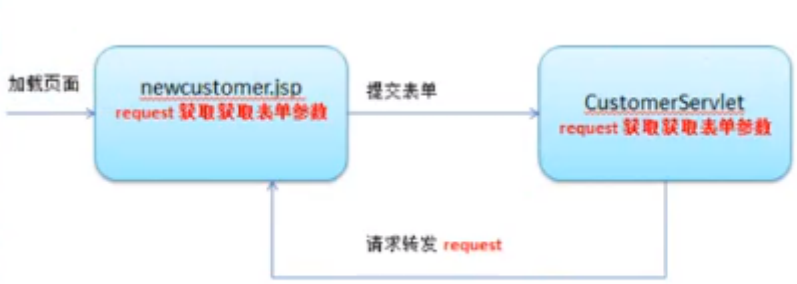
删除操作

删除操作完成后要重定向到query.do

```
1  private void delete(HttpServletRequest req, HttpServletResponse response)
throws IOException {
2   String strId = req.getParameter("id");
3   int id = 0;
4   //防止id转换错误,如果错误仍然执行query.do
5   try {
6   id = Integer.parseInt(strId);
7   customerDao.delete(id);
```

```
8   } catch (Exception e) {
9   }
10   response.sendRedirect("query.do");
11  }
```

增加新Customer操作




```
1   private void addCustomer(HttpServletRequest req, HttpServletResponse res
p) throws ServletException, IOException {
2   //1.获取表单参数
3   String name = req.getParameter("name");
4   String address = req.getParameter("address");
5   String phone = req.getParameter("phone");
6
7   //2.查看名字是否被占用
8   //2.1 getCountWithName()来查看
9   int count = (int) customerDao.getCountWithName(name);
10
11   //2.2 如果值大于0 转发响应newCustomer.jsp，需要保持原数据信息，并显示错误消
息
12  // req.getRequestDispatcher("/jsp/addCustomer.jsp");
13   if (count > 0) {
14   req.setAttribute("message","用户名"+name+"已被占用，请重新选择");
15   req.getRequestDispatcher("addCustomer.jsp").forward(req, resp);
16   return;
17   }
18
19   Customer customer = new Customer(name,address,phone);
20   customerDao.save(customer);
21   //3.如果不存在，则添加Customer，并回到添加成功界面
22   //因为在当前路径 /jsp，所以直接使用success
23   resp.sendRedirect("success.jsp");
24   }
```

这里要注意一个问题就是，页面转发时的路径url问题

修改数据

隐藏域

使用隐藏域来保存要修改的customer对象的id，可以和其他表单域一样被提交到服务器，只不过页面上不显示。

```
<%--使用隐藏域--%>
<input type="hidden" name="id" value="<%=id%>">
```

```
private void update(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
 //1.获取表单数据
 String id = request.getParameter("id");
 String name = request.getParameter("name");
 String oldName = request.getParameter("oldName");
 String address = request.getParameter("address");
 String phone = request.getParameter("phone");

 //2.检查name是否已经存在
 if (!name.equals(oldName)) {
 long count = customerDao.getCountWithName(name);
 //检查数据库中时否已经存在
 if (count > 0) {
 request.setAttribute("message", "用户" + name + "已经存在");
 //转发到原来的界面
 request.getRequestDispatcher("jsp/update.jsp").forward(request, response);
 return;
 }

 //数据库中不存在，就更新用户
 Customer customer = new Customer(name, phone, address);
 customer.setId(Integer.parseInt(id));
 customerDao.update(customer);

 response.sendRedirect("query.do");
 }
```

```
27 }
```