## 一、引言

Mybatis提供了强大的分页拦截实现，可以完美的实现分功能

## 二、普通的分页实现

普通分页实现即使直接在mapper文件中写分页查询语句

Messsage.xml

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE mapper
3. PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4. "http://mybatis.org/dtd/mybatis-3-mapper.dtd"**>**
6. **<mapper** namespace="com.imooc.bean.Message"**>**
8. **<resultMap** type="com.imooc.bean.Message" id="MessageResult"**>**
9. **<id** column="id" jdbcType="INTEGER" property="id"**/>**
10. **<result** column="command" jdbcType="VARCHAR" property="command"**/>**
11. **<result** column="description" jdbcType="VARCHAR" property="description"**/>**
12. **<result** column="content" jdbcType="VARCHAR" property="content"**/>**
13. **</resultMap>**
15. <!--
16. 普通分页实现
17. 使用sql实现分页查询，不是使用拦截器
18. oracle分页使用rownum,子查询呢需要使用别名不能使用rownum，row等关键字
19. --**>**
20. **<select** id="queryMessageListByPage" parameterType="java.util.Map" resultMap="MessageResult"**>**
21. select \* from ( select rownum r, id,command,description,content from message
22. **<where>**
23. **<if** test="command != null and command.trim() != ''"**>**
24. and command=#{command}
25. **</if>**
26. **<if** test="descriptioin != null and description.trim() != '' "**>**
27. and descriptioin=#{descriptioin}
28. **</if>**
29. **<if** test="page != null"**>**
30. and rownum **<**= #{page.dbNumber}
31. **</if>**
32. **</where>**
33. )
34. **<where>**
35. and r **>** #{page.dbIndex}
36. **</where>**
37. order by id
38. **</select>**
39. **<select** id="count" parameterType="com.imooc.bean.Message"  resultType="int"**>**
40. select count(\*) from message
41. **<where>**
42. **<if** test="command != null and !"".equals(command.trim())"**>**
43. and command=#{command}
44. **</if>**
45. **<if** test="description != null and ''!=description.trim()"**>**
46. and description like concat(concat('%',#{description}),'%')
47. **</if>**
48. **</where>**
49. **</select>**
50. **</mapper>**

Page.**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "/home/caojx/文档\\x/_blank)**分页相关的参数都通过该类对象设置

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. **package** com.imooc.common;
3. /\*\*
4. \* 分页对应的实体类
5. \*/
6. **public** **class** Page {
7. /\*\*
8. \* 总条数
9. \* \*/
10. **private** **int** totalNumber;
12. /\*\*
13. \* 总页数
14. \* \*/
15. **private** **int** totalPage;
17. /\*\*
18. \* 当前页
19. \* \*/
20. **private** **int** currentPage;
22. /\*\*
23. \* 每页显示的数目
24. \* \*/
25. **private** **int** pageNumber = 5;
27. /\*\*
28. \* 数据库中limit的参数，从第几条开始取
29. \* \*/
30. **private** **int** dbIndex;
32. /\*\*
33. \* 数据库中limit的参数，一共取多少条,适用于mysql, 如果是oracle则表示最大取到的条数
34. \* \*/
35. **private** **int** dbNumber;
37. /\*\*
38. \* 根据当前对象中的属性值计算并设置相关属性的值
39. \* \*/
41. **public** **void** count() {
42. // 计算总页数
43. **int** totalPageTemp = **this**.totalNumber / **this**.pageNumber;
44. **int** plus = (**this**.totalNumber % **this**.pageNumber) == 0 ? 0 : 1;
45. totalPageTemp += plus;
46. // 如果总页数小于0显示第一页
47. **if** (totalPageTemp <= 0) {
48. totalPageTemp += 1;
49. }
50. **this**.totalPage = totalPageTemp;
51. //设置limit参数
52. **this**.dbIndex = (**this**.currentPage -1 ) \* **this**.pageNumber;
53. **this**.dbNumber = **this**.pageNumber;
54. }
56. **public** **int** getTotalNumber() {
57. **return** totalNumber;
58. }
60. **public** **void** setTotalNumber(**int** totalNumber) {
61. **this**.totalNumber = totalNumber;
62. count();
63. }
65. **public** **int** gettotalPage() {
66. **return** totalPage;
67. }
69. **public** **void** settotalPage(**int** totalPage) {
70. **this**.totalPage = totalPage;
71. }
73. **public** **int** getCurrentPage() {
74. **return** currentPage;
75. }
77. **public** **void** setCurrentPage(**int** currentPage) {
78. **this**.currentPage = currentPage;
79. count();
80. }
82. **public** **int** getPageNumber() {
83. **return** pageNumber;
84. }
86. **public** **void** setPageNumber(**int** pageNumber) {
87. **this**.pageNumber = pageNumber;
88. }
90. **public** **int** getDbIndex() {
91. **return** dbIndex;
92. }
94. **public** **void** setDbIndex(**int** dbIndex) {
95. **this**.dbIndex = dbIndex;
96. }
98. **public** **int** getDbNumber() {
99. **return** dbNumber;
100. }
102. **public** **void** setDbNumber(**int** dbNumber) {
103. **this**.dbNumber = dbNumber;
104. }
106. }

Dao层实现

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. **package** com.imooc.dao;
3. **import** java.util.HashMap;
4. **import** java.util.List;
5. **import** java.util.Map;
7. **import** org.apache.ibatis.session.SqlSession;
8. **import** com.imooc.bean.Message;
9. **import** com.imooc.common.Page;
10. **import** com.imooc.db.DBAccess;
12. **public** **class** MessageDao {
14. DBAccess dbAccess = **new** DBAccess();
16. //计算查询的数据的总数
18. **public** **int** getMessageCount(String command,String description) **throws** Exception{
19. **int** count = 0;
20. SqlSession session = **null**;
21. **try** {
22. session = dbAccess.getSqlSession();
23. Message message = **new** Message();
24. message.setCommand(command);
25. message.setDescription(description);
26. count = session.selectOne(Message.**class**.getName()+".count", message);
27. } **catch** (Exception e) {
28. **throw** **new** Exception(e.getMessage());
29. } **finally** {
30. **if** (session != **null**) {
31. session.close();
32. }
33. }
34. **return** count;
35. }
37. //分页实现主要的方法
38. **public** List<Message> queryMessageListByPage(String command, String description,Page page) **throws** Exception {
39. List<Message> messagesList = **null**;
40. SqlSession session = **null**;
41. **try** {
42. session = dbAccess.getSqlSession();
43. Map<String, Object> paramater = **new** HashMap<String, Object>();
44. paramater.put("command", command);
45. paramater.put("description", description);
46. paramater.put("page", page);
47. messagesList = session.selectList(Message.**class**.getName()+".queryMessageListByPage2", paramater);
48. } **catch** (Exception e) {
49. e.printStackTrace();
50. **throw** **new** Exception(e.getMessage());
51. } **finally** {
52. **if** (session != **null**) {
53. session.close();
54. }
55. }
56. **return** messagesList;
57. }

60. }

Service层实现

调用Dao层的方法，返回本次查询的数据

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. package com.imooc.service;
3. import java.util.ArrayList;
4. import java.util.List;
5. import com.imooc.bean.Message;
6. import com.imooc.common.Page;
7. import com.imooc.dao.MessageDao;
9. public class MessageService {
11. MessageDao dao = new MessageDao();
13. public int getCount(String command, String description) throws Exception {
14. int count = 0;
15. try {
16. count= dao.getMessageCount(command, description);
17. } catch (Exception e) {
18. throw new Exception(e.getMessage());
19. }
20. return count;
21. }

24. public List**<Message>** queryMessageListByPage(String command, String description,Page page) throws Exception {
25. List**<Message>** messageList = null;
26. try {
27. messageList = dao.queryMessageListByPage(command, description, page);
28. } catch (Exception e) {
29. throw new Exception(e.getMessage());
31. }
32. return messageList;
33. }
34. }

ListServet.java

用于接收前台页面的参数，以及传回数据给jsp页面

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. **package** com.imooc.servlet;
3. **import** java.io.IOException;
4. **import** java.util.List;
5. **import** java.util.regex.Pattern;
7. **import** javax.servlet.ServletException;
8. **import** javax.servlet.http.HttpServlet;
9. **import** javax.servlet.http.HttpServletRequest;
10. **import** javax.servlet.http.HttpServletResponse;
12. **import** com.imooc.bean.Message;
13. **import** com.imooc.common.Page;
14. **import** com.imooc.service.MessageService;
16. **public** **class** ListServlet **extends** HttpServlet {
18. /\*\*
19. \*
20. \*/
21. **private** **static** **final** **long** serialVersionUID = 1L;
22. MessageService messageService = **new** MessageService();
24. @Override
25. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) {
26. **try** {
27. // 设置编码
28. req.setCharacterEncoding("utf-8");
29. // 接受参数
30. String command = req.getParameter("command");
31. String description = req.getParameter("description");
32. String currentPage = req.getParameter("currentPage");
33. //创建分页对象
34. Page page = **new** Page();
35. //设置总条数
36. page.setTotalNumber(messageService.getCount(command, description));
37. Pattern patttern = Pattern.compile("[0-9]{1,9}");
38. **if**(currentPage == **null** || !patttern.matcher(currentPage).matches()){
39. page.setCurrentPage(1);
40. }**else** {
41. page.setCurrentPage(Integer.parseInt(currentPage));
42. }
43. //oracle数据库分页，dbNumber表示最大能取到的条数
44. page.setDbNumber(page.getDbIndex()+page.getPageNumber());
45. List<Message> messageList = messageService.queryMessageListByPage(command, description, page);
46. req.setAttribute("command", command);
47. req.setAttribute("description", description);
48. req.setAttribute("messages", messageList);
49. req.setAttribute("page", page);
50. // 向页面跳转
51. req.getRequestDispatcher("/WEB-INF/jsp/back/list.jsp").forward(req,resp);
52. } **catch** (Exception e) {
53. e.printStackTrace();
54. req.setAttribute("retMsg", "查询失败");
55. }
56. }
58. @Override
59. **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp)
60. **throws** ServletException, IOException {
61. doGet(req, resp);
62. }
64. }

jsp页面实现

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"**>**
2. **<**%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %**>**
3. **<**%
4. String path = request.getContextPath();
5. String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
6. %**>**
7. **<html** xmlns="http://www.w3.org/1999/xhtml"**>**
8. **<head>**
9. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8" **/>**
10. **<meta** http-equiv="X-UA-Compatible"content="IE=9; IE=8; IE=7; IE=EDGE" **/>**
11. **<title>**内容列表页面**</title>**
12. **<link** href="<%= basePath %>css/all.css" rel="stylesheet" type="text/css" **/>**
13. **<script** src="<%= basePath %>js/common/jquery-1.8.0.min.js"**></script>**
14. **<script** src="<%= basePath %>js/back/list.js"**></script>**
15. **</head>**
16. **<body** style="background: #e1e9eb;"**>**
17. **<form** action="<%= basePath %>List.action" id="mainForm" method="post"**>**
18. **<input** type="hidden" name="currentPage" id="currentPage" value="${page.currentPage}"**/>**
20. **<div** class="right"**>**
21. **<div** class="current"**>**当前位置：**<a** href="javascript:void(0)" style="color:#6E6E6E;"**>**内容管理**</a>** **>** 内容列表**</div>**
22. **<div** class="rightCont"**>**
23. **<p** class="g\_title fix"**>**内容列表 **<a** class="btn03" href="#"**>**新 增**</a>**    **<a** class="btn03" href="javascript:deleteBatch('<%=basePath%>');"**>**删 除**</a></p>**
24. **<table** class="tab1"**>**
25. **<tbody>**
26. **<tr>**
27. **<td** width="90" align="right"**>**指令名称：**</td>**
28. **<td>**
29. **<input** name="command" type="text" class="allInput" value="${command}"**/>**
30. **</td>**
31. **<td** width="90" align="right"**>**描述：**</td>**
32. **<td>**
33. **<input** name="description" type="text" class="allInput" value="${description}"**/>**
34. **</td>**
35. **<td** width="85" align="right"**><input** type="submit" class="tabSub" value="查 询" **/></td>**
36. **</tr>**
37. **</tbody>**
38. **</table>**
39. **<div** class="zixun fix"**>**
40. **<table** class="tab2" width="100%"**>**
41. **<tbody>**
42. **<tr>**
43. **<th><input** type="checkbox" id="all" onclick="javascript:selAllCheckbox('id')"**/></th>**
44. **<th>**序号**</th>**
45. **<th>**指令名称**</th>**
46. **<th>**描述**</th>**
47. **<th>**操作**</th>**
48. **</tr>**
49. **<c:forEach** items="${requestScope.messages}" var="message" varStatus="status"**>**
50. **<tr**  **<c:if** test="${status.index % 2 != 0}"**>**style='background-color:#ECF6EE;'**</c:if>>**
51. **<td><input** type="checkbox"  name="id" value="${message.id}"**/></td>**
52. **<td>**${status.index + 1}**</td>**
53. **<td>**${message.command}**</td>**
54. **<td>**${message.description}**</td>**
55. **<td>**
56. **<a** href="#"**>**修改**</a>**
57. **<a** href="${basePath}DeleteOneServlet.action?id=${message.id}"**>**删除**</a>**
58. **</td>**
59. **</tr>**
60. **</c:forEach>**
61. **</tbody>**
62. **</table>**
63. **<div** class='page fix'**>**
64. 共 **<b>**${page.totalNumber}**</b>** 条
65. 共**<b>**${page.totalPage }**</b>**页
66. **<c:if** test="${page.currentPage != 1}"**>**
67. **<a** href="javascript:changeCurrentPage('1')" class='first'**>**首页**</a>**
68. **<a** href="javascript:changeCurrentPage('${page.currentPage-1}')" class='pre'**>**上一页**</a>**
69. **</c:if>**
70. 当前第**<span>**${page.currentPage}/${page.totalPage}**</span>**页
71. **<c:if** test="${page.currentPage != page.totalPage}"**>**
72. **<a** href="javascript:changeCurrentPage('${page.currentPage+1}')" class='next'**>**下一页**</a>**
73. **<a** href="javascript:changeCurrentPage('${page.totalPage}')" class='last'**>**末页**</a>**
74. **</c:if>**
75. 跳至 **<input** id="currentPageText" type='text' value='${page.currentPage}' class='allInput w28' **/>** 页
76. **<a** href="javascript:changeCurrentPage($('#currentPageText').val())" class='go'**>**GO**</a>**
77. **</div>**
78. **</div>**
79. **</div>**
80. **</div>**
81. **</form>**
82. **</body>**
83. **</html>**

使用jquery提交分页参数

/\*  
 \* 实现翻页功能  
 \*/  
function changeCurrentPage(currentPage){  
$("#currentPage").val(currentPage).val();  
$("#mainForm").submit();  
}

## 三、使用Mybatis分页拦截器实现

 使用Mybatis分页拦截器，我们可以不用在Mapper配置文件中写分页查询语句，我们只需要写非分页查询语句就行，然后通过分页拦截器，拦截到需要分页查询的普通sql

将普通的sql替换成分页sql，非常巧妙的实现分页查询

1.实现拦截器我们需实现Interceptor

我们假如将拦截器形象的比喻成够票与代理商

a.@Intercepts注解声明了代够员取哪里拦截需要买票的顾客（去哪里拦截顾客）

b.setProperties获取Configuration.xml中plugins--》plugin--->propertis中的固定资本（公司固定资本）

c.plugin方法中判断是否需要拦截（顾客是否需要购票）

d.intercept方法中可以获取原始的sql(非分页查询的sql),和分页参数   通过取到这些参数替换掉原来的sql即编程分页sql(代购员从顾客哪里获取信息)

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. **package** com.imooc.interceptor;
3. **import** java.sql.Connection;
4. **import** java.sql.PreparedStatement;
5. **import** java.sql.ResultSet;
6. **import** java.util.Map;
7. **import** java.util.Properties;
9. **import** org.apache.ibatis.executor.parameter.ParameterHandler;
10. **import** org.apache.ibatis.executor.statement.RoutingStatementHandler;
11. **import** org.apache.ibatis.executor.statement.StatementHandler;
12. **import** org.apache.ibatis.mapping.BoundSql;
13. **import** org.apache.ibatis.mapping.MappedStatement;
14. **import** org.apache.ibatis.plugin.Interceptor;
15. **import** org.apache.ibatis.plugin.Intercepts;
16. **import** org.apache.ibatis.plugin.Invocation;
17. **import** org.apache.ibatis.plugin.Plugin;
18. **import** org.apache.ibatis.plugin.Signature;
19. **import** org.apache.ibatis.reflection.MetaObject;
20. **import** org.apache.ibatis.reflection.SystemMetaObject;
21. **import** org.apache.ibatis.session.Configuration;
23. **import** com.imooc.common.Page;
25. /\*\*
26. \* myBatis分页拦截器实现
27. \* 拦截器在配置文件Configuration.xml中注册后，通过下边的注解声明拦截的类，方法，方法参数
28. \* \*/
29. //type指向需要拦截的接口，method指向需要拦截的方法，args指向方法中的参数
30. //StatementHandler是一个接口，实现类是BaseStatementHanlder，实现了其中的prepare方法，实现方法中instantiateStatement(connection)
31. //返回Statement
32. @Intercepts({@Signature(type=StatementHandler.**class**,method="prepare",args={Connection.**class**})})
33. **public** **class** PageInterceptor **implements** Interceptor{
34. **private** Object object;
35. **private** **static** String defualtSqlId = ".+ByPage$";  //需要拦截的配置文件中的id的正则表达式
36. **private** **static** String defaultDialect = "oracle";   //默认数据库类型
37. **private** String dialect;                 //数据库类型
39. /\*\*
40. \* 3.代购
41. \* 可以指定拦截映射文件中那种id的值，
42. \* invocation中可以获取想要的数据
43. \* \*/
44. @Override
45. **public** Object intercept(Invocation invocation) **throws** Throwable {
46. //代理业务员获取代理对象的携带的信息（需要买票的人哪里获取信息）
47. StatementHandler statementHandler = (StatementHandler)invocation.getTarget();
48. //使用metaObject中获取statementHandler属性的值
49. MetaObject metaObject = MetaObject.forObject(statementHandler, SystemMetaObject.DEFAULT\_OBJECT\_FACTORY, SystemMetaObject.DEFAULT\_OBJECT\_WRAPPER\_FACTORY);
50. //BaseStatementHandler-->mappedStatement(delegate为<span style="font-family: Arial, Helvetica, sans-serif;">mappedStatement属性名</span><span style="font-family: Arial, Helvetica, sans-serif;">)</span>
51. MappedStatement mappedStatement=(MappedStatement) metaObject.getValue("delegate.mappedStatement");
52. Configuration configuration = (Configuration) metaObject.getValue("delegate.configuration");
53. //获取数据库类型，获取Configuration.xml properties标签中声明的变量
54. dialect = configuration.getVariables().getProperty("dialect");
55. **if** (**null** == dialect || "".equals(dialect)) {
56. dialect = defaultDialect;
57. }
58. //获取mapper文件中sql语句的id
59. String pageSqlId = mappedStatement.getId();
60. **if**(pageSqlId.matches(defualtSqlId)){//获取已Bypage结尾的id中的sql语句         拦截谁
61. BoundSql boundSql = statementHandler.getBoundSql();
62. //获取原始的sql
63. String sql = boundSql.getSql();
64. //查询总条数
65. String countSql = "select count(\*) from ("+sql+") a";
66. Connection connection = (Connection) invocation.getArgs()[0];
67. PreparedStatement countStatement = connection.prepareStatement(countSql);
68. //获取原始的sql参数信息
69. ParameterHandler parameterHandler =  (ParameterHandler) metaObject.getValue("delegate.parameterHandler");
70. //设置参数值
71. parameterHandler.setParameters(countStatement);
72. //执行获取总条数
73. ResultSet rs = countStatement.executeQuery();
75. //改造成带分页查询的sql，获取传给配置文件的参数,就是MessageDao中queryMessageListByPage中方法执行中传递进去的map参数，所以需要强转
76. Map<String, Object> parameter = (Map<String, Object>) boundSql.getParameterObject();
77. //paramter(map)中传入的page对象的参数
78. Page page = (Page)parameter.get("page");
79. //设置page的总条数
80. **if**(rs.next()){
81. page.setTotalNumber(rs.getInt(1));
82. }
83. //改造成带分页查询的sql
84. String pageSql = buildPageSql(sql,page);
85. //替换员来的sql
86. //第一参数：找到源码中的sql
87. //第二个参数：改造后的sql
88. metaObject.setValue("delegate.boundSql.sql", pageSql);          //买票
89. }
90. **return** invocation.proceed();   //交回主权，买票后送回
91. }
93. /\*\*
94. \* 2.定位客户  @Intercepts({@Signature(type=StatementHandler.class,method="prepare",args={Connection.class})})
95. \* 如果拦截成功，会返回一个代理类（业务员），然后会执行intercept方法
96. \* 就好比拦截到买票的人，需不需要买票，不需要放行，return就是放行
97. \* 需要就代表一种协议Plugin.wrap(target, this);
98. \* target就好比拦截住的需要买票的人,this就是公司的业务员，返回后this就变成可以代理target去买票的业务员
99. \* \*/
100. @Override
101. **public** Object plugin(Object target) {
102. **return** Plugin.wrap(target, **this**);
103. }
105. /\*\*
106. \* 1.获取固定资本
107. \* 获取Configuration.xml中的plugin中的property中的属性值
108. \* \*/
109. @Override
110. **public** **void** setProperties(Properties properties) {
111. **this**.object = properties.get("test");
112. }
114. /\*\*
115. \* 改造成带分页查询的sql
116. \* sql原始sql(不带分页功能)
117. \* Page page分页实体类对象
118. \*\*/
119. **public** String buildPageSql(String sql,Page page){
120. StringBuilder builder = **new** StringBuilder();
121. **if**(dialect.equals("oracle")){
122. builder = pageSqlForOracle(sql, page);
123. }
124. **if**(dialect.equals("mysql")){
125. pageSqlForMysql(sql, page);
126. }
127. **return** builder.toString();
128. }
129. //mysql分页查询
130. **public** StringBuilder pageSqlForMysql(String sql,Page page){
131. StringBuilder builder = **new** StringBuilder();
132. builder.append(sql);
133. builder.append(" limit "+page.getDbIndex()+","+page.getDbNumber());
134. builder.append(" order by id");
135. **return** builder;
136. }
138. /\*\*
139. \* 实现oracle分页
140. \* \*/
141. **public** StringBuilder pageSqlForOracle(String sql,Page page){
142. StringBuilder builder = **new** StringBuilder();
143. builder.append("select \* from ( select rownum r, id,command,description,content from (");
144. builder.append(sql);
145. builder.append(") where rownum <= ").append(page.getDbNumber()+page.getDbIndex());
146. builder.append(") where r > ").append(page.getDbIndex());
147. builder.append(" order by id");
148. System.out.println(builder.toString());
149. **return** builder;
150. }
151. }

Configuration.xml中注册分页拦截器插件

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. **<?xml** version="1.0" encoding="UTF-8" **?>**
2. <!DOCTYPE configuration
3. PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4. "http://mybatis.org/dtd/mybatis-3-config.dtd"**>**
6. **<configuration>**
7. **<properties>**
8. **<property** name="dialect" value="oracle" **/>**
9. **</properties>**
10. <!-- 拦截器可以有很多，就像代购公司有很多每一个公司都需要注册，下边就是注册 -->
11. **<plugins>**
12. **<plugin** interceptor="com.imooc.interceptor.PageInterceptor"**>**
13. **<property** name="test" value="interceptor" **/>**<!-- property中的值可以在拦截器中的setProperties中拿到 -->
14. **</plugin>**
15. **</plugins>**
17. **<environments** default="development"**>**
18. **<environment** id="development"**>**
19. **<transactionManager** type="JDBC"**>**
20. **<property** name="" value="" **/>**
21. **</transactionManager>**
22. **<dataSource** type="UNPOOLED"**>**
23. **<property** name="driver" value="oracle.jdbc.driver.OracleDriver" **/>**
24. <!-- <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/> -->
25. **<property** name="url" value="jdbc:oracle:thin:@localhost:1521:xe" **/>**
26. **<property** name="username" value="caojx" **/>**
27. **<property** name="password" value="caojx" **/>**
28. **</dataSource>**
29. **</environment>**
30. **</environments>**
32. **<mappers>**
33. **<mapper** resource="com/imooc/config/sqlxml/Message.xml" **/>**
34. **</mappers>**
36. **</configuration>**

Message.xml变化

不需要写分语句

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
3. <!DOCTYPE mapper
4. PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
5. "http://mybatis.org/dtd/mybatis-3-mapper.dtd"**>**
7. **<mapper** namespace="com.imooc.bean.Message"**>**
9. **<resultMap** type="com.imooc.bean.Message" id="MessageResult"**>**
10. **<id** column="id" jdbcType="INTEGER" property="id"**/>**
11. **<result** column="command" jdbcType="VARCHAR" property="command"**/>**
12. **<result** column="description" jdbcType="VARCHAR" property="description"**/>**
13. **<result** column="content" jdbcType="VARCHAR" property="content"**/>**
14. **</resultMap>**
16. <!-- select标签用于书写查询语句-->
17. **<select** id="queryMessageListByPage" parameterType="java.util.Map" resultMap="MessageResult"**>**
18. select id,command,description,content from message
19. **<where>**
20. **<if** test="command != null and !"".equals(command.trim())"**>**
21. and command=#{command}
22. **</if>**
23. <!-- like 查询一般会拼接concat()拼接两个字符串 -->
24. **<if** test="description != null and ''!=description.trim()"**>**
25. and description like concat(concat('%',#{description}),'%')
26. **</if>**
27. **</where>**
28. **</select>**
29. **</mapper>**

ListServlet.java中的一小点变化

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/52089808" \o "view plain) [copy](http://blog.csdn.net/u014292162/article/details/52089808" \o "copy)

1. **package** com.imooc.servlet;
3. **import** java.io.IOException;
4. **import** java.util.List;
5. **import** java.util.regex.Pattern;
7. **import** javax.servlet.ServletException;
8. **import** javax.servlet.http.HttpServlet;
9. **import** javax.servlet.http.HttpServletRequest;
10. **import** javax.servlet.http.HttpServletResponse;
12. **import** com.imooc.bean.Message;
13. **import** com.imooc.common.Page;
14. **import** com.imooc.service.MessageService;
16. **public** **class** ListServlet **extends** HttpServlet {
18. /\*\*
19. \*
20. \*/
21. **private** **static** **final** **long** serialVersionUID = 1L;
22. MessageService messageService = **new** MessageService();
24. @Override
25. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) {
26. **try** {
27. // 设置编码
28. req.setCharacterEncoding("utf-8");
29. // 接受参数
30. String command = req.getParameter("command");
31. String description = req.getParameter("description");
32. String currentPage = req.getParameter("currentPage");
33. //创建分页对象
34. Page page = **new** Page();
35. Pattern patttern = Pattern.compile("[0-9]{1,9}");
36. **if**(currentPage == **null** || !patttern.matcher(currentPage).matches()){
37. page.setCurrentPage(1);
38. }**else** {
39. page.setCurrentPage(Integer.parseInt(currentPage));
40. }
41. List<Message> messageList = messageService.queryMessageListByPage(command, description, page);
42. req.setAttribute("command", command);
43. req.setAttribute("description", description);
44. req.setAttribute("messages", messageList);
45. req.setAttribute("page", page);
46. // 向页面跳转
47. req.getRequestDispatcher("/WEB-INF/jsp/back/list.jsp").forward(req,resp);
48. } **catch** (Exception e) {
49. e.printStackTrace();
50. req.setAttribute("retMsg", "查询失败");
51. }
52. }
54. @Override
55. **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp)
56. **throws** ServletException, IOException {
57. doGet(req, resp);
58. }
60. }

结果图



注：本案例思想有慕课网提供指导