过滤查询

一、过滤查询

过滤查询实则是根据用户传入的条件进行数据的筛选查询，最终返回结果给用户。

1、需求及效果图

例如：在商品列表中，用户可以根据自己的需要，按照商品的名称和价格范围对商品进行查询。

货品名: M 价格: 100 - 500 查询

编号	货品名	分类编号	零售价	供应商	品牌	折扣	进货价
1	罗技M90	3	90.00	罗技	罗技	0.5	35.00
2	罗技M100	3	49.00	罗技	罗技	0.9	33.00
3	罗技M115	3	99.00	罗技	罗技	0.6	38.00

首页 上一页 下一页 尾页 当前第 1 / 193 页 一共 578 条数据 跳转到 1 页 每页显示 3 条数据

2、原理分析

以上条件设计，后台需执行的 SQL 会有如下几种：

- 需求 1：查询所有货品信息。

```
SELECT * FROM product
```

- 需求 2：查询货品名称包含某个字符串的货品信息，例如包含 M 的。

```
SELECT * FROM product WHERE productName LIKE '%M%'
```

- 需求 3：查询货品零售价在某个价格以下的货品信息，例如销售价小于等于 500。

```
SELECT * FROM product WHERE salePrice <= 500
```

- 需求 4：查询货品零售价在某个价格以下的货品信息，例如销售价大于等于 200。

```
SELECT * FROM product WHERE salePrice >= 200
```

- 需求 5：查询零售价在某个区间的的货品信息，例如且销售价在 200 到 500 之间（包含）。

```
SELECT * FROM product WHERE salePrice >= 200 AND salePrice <= 500
```

- 需求 6：查询货品名称包含某个字符串且零售价在某个价格以上的货品信息，例如名称包含 M，且销售价大于等于 200。

```
SELECT * FROM product WHERE productName LIKE '%M%' AND salePrice >= 200
```

- 需求 7：查询货品名称包含某个字符串且零售价在某个价格以下的货品信息，例如名称包含 M，且销售价小于等于 500。

```
SELECT * FROM product WHERE productName LIKE '%M%' AND salePrice <= 500
```

- 需求 8：查询货品名称包含某个字符串且零售价在某个区间的的货品信息，例如名称包含 M，且销售价在 200 到 500 之间（包含）。

```
SELECT * FROM product WHERE productName LIKE '%M%' AND  salePrice >= 200 AND
salePrice <= 500
```

从以上 SQL 分析，不管如何查询，SELECT * FROM product 这部分 SQL 是固定不变，变的地方就只是 WHERE 子句部分，而这个子句的变化，取决于用户在页面传递的查询参数。因此，**过滤查询说直白点就是根据用户查询传递的参数，拼接 WHERE 条件，这就是过滤查询的底层原理。**

3、需解决的问题

- 如何封装用户传递到后台的请求参数（过滤条件数据）？
- 如何根据用户传递的参数，在 Mapper XML 中拼接对应的 SQL?

解决方案：第一个问题定义一个类封装数据即可。第二个问题使用 MyBaits 的动态 SQL 来解决。

二、过滤条件数据封装

货品名: M 价格: 100 - 500 查询

编号	货品名	分类编号	零售价	供应商	品牌	折扣	进货价
1	罗技M90	3	90.00	罗技	罗技	0.5	35.00
2	罗技M100	3	49.00	罗技	罗技	0.9	33.00
3	罗技M115	3	99.00	罗技	罗技	0.6	38.00

首页 上一页 下一页 尾页 当前第 1 / 193 页 一共 578 条数据 跳转到 1 页 每页显示 3 条数据

根据前面我们对多条件过滤查询的分析得知，用户过滤查询时，请求时会传递可能会传递多个过滤条件的参数来查询对应的数据，又因为这些参数需要在后台多个层次之间进行传递（servlet--->service--->dao），所以为了方便参数传递，我们选择将这些参数封装到指定的对象中，然后再在多层之间进行传递。

编写 ProductQueryObject.java

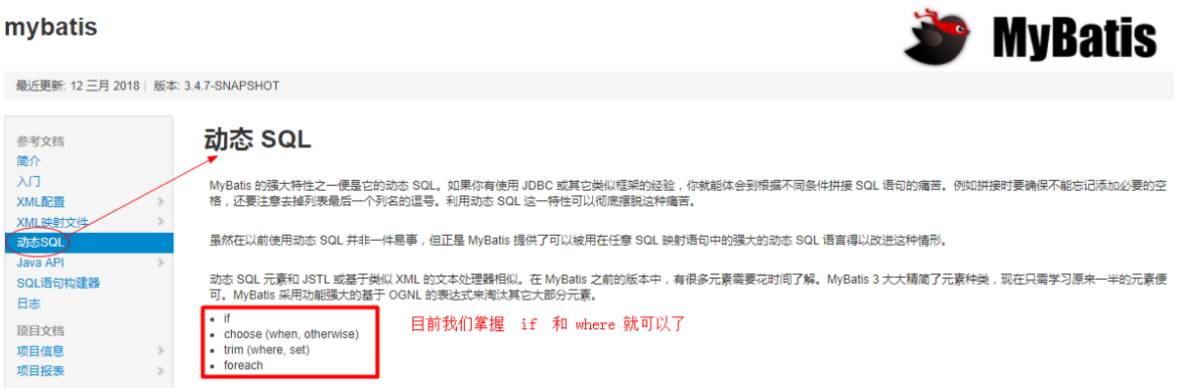
用来封装过滤查询的参数。但由于产品查询又想支持分页查询，又想支持过滤查询，所以使用 ProductQueryObject 来 继承 QueryObject，这样既可以封装分页查询的参数，又可以封装过滤查询的参数，且减少代码重复。

```
package cn.wolfcode.qo;

@Setter
@Getter
public class ProductQueryObject extends QueryObject {
    private String productName;         // 商品名
    private BigDecimal minSalePrice;    // 最小销售价格
    private BigDecimal maxSalePrice;    // 最大销售价格
}
```

三、MyBatis 动态 SQL

为了解决在 Mapper XML 中拼接 SQL 的问题，此处需要来学习 MyBatis 的动态 SQL.



1、if 标签

1.1、语法与作用

用于单条件判断，一般用于判断是否符合某一查询条件。语法如下：

```
<if test="boolean 表达式"></if>
```

boolean 表达式：

- 可以写类似这样（productName != null）booolean 表达式。
- 表达式中可以使用逻辑运算符，使用小写 and，or，但不可用 &&，||。

1.2、需求应用

需求：查询货品名称包含某个字符且零售价在某个区间的所有货品信息，商品名、最小销售价格和最大销售价格可能传也可能没传。

```
<select id="queryForList" resultType="cn.wolfcode.domain.Product">
    SELECT * FROM product
    <if test="productName != null and productName != ''">
        WHERE productName LIKE concat('%', #{productName}, '%')
    </if>
    <if test="minSalePrice != null">
        AND salePrice >= #{minSalePrice}
    </if>
    <if test="maxSalePrice != null">
        AND salePrice <= #{maxSalePrice}
    </if>
    LIMIT #{start}, #{pageSize}
</select>
```

1.3、存在的问题

- minSalePrice 和 maxSalePrice 条件是可选的，可能传也可能没传，此时造成拼接的 SQL 出现语法问题.
- < 报错，因为使用了 XML 的标签字符。

解决方案：第一个问题使用 where 标签解决，第二个问题使用转义符解决。

2、where 标签

2.1、语法与作用

用于在第一个条件语句之前加入 WHERE 关键字或者去除多余的 AND 或 OR 关键字，语法如下：

```
<where>条件语句</where>
```

注意：
若查询条件语句没有 WHERE 关键字，则自动在查询条件语句之前插入 WHERE；
若查询条件语句以 "AND" 或 "OR" 开头，则把第一个 AND 或 OR 使用 WHERE 关键字替换。

2.2、需求应用

需求：查询货品名称包含某个字符且零售价在某个区间的所有货品信息，关键字、最小销售价格和最大销售价格可能传也可能没传。

```
<select id="queryForList" resultType="cn.wolfcode.domain.Product">
    SELECT * FROM product
    <where>
        <if test="productName != null and productName != ''">
            AND productName LIKE concat('%', #{productName}, '%')
        </if>
        <if test="minSalePrice != null">
            AND salePrice >= #{minSalePrice}
        </if>
        <if test="maxSalePrice != null">
            AND salePrice <= #{maxSalePrice}
        </if>
    </where>
    LIMIT #{start}, #{pageSize}
</select>
```


3、MyBatis 转义符号

```
&lt;    <  小于号
&gt;    >  大于号
&amp;   &  与符号
&apos;  '  单引号
&quot;  "  双引号
```

```
<select id="queryForList" resultType="cn.wolfcode.domain.Product">
    SELECT * FROM product
    <where>
        <if test="productName != null and productName != ''">
            AND productName LIKE concat('%', #{productName}, '%')
        </if>
        <if test="minSalePrice != null">
            AND salePrice &gt;= #{minSalePrice}
        </if>
        <if test="maxSalePrice != null">
            AND salePrice &lt;= #{maxSalePrice}
        </if>
```

```
    </where>
    LIMIT #{start}, #{pageSize}
</select>
```

四、过滤查询实现

需求：查询货品名称包含某个字符串且零售价在某个区间的所有货品信息。

1、编写 ProductQueryObject.java

继承 QueryObject，用于封装查询产品过滤查询的参数值。

```
package cn.wolfcode.qo;

@Setter
@Getter
public class ProductQueryObject extends QueryObject {
    private String productName;         // 商品名
    private BigDecimal minSalePrice;    // 最小销售价格
    private BigDecimal maxSalePrice;    // 最大销售价格
}
```

注意：一般的为了避免不必要的麻烦，让字段名称和表单中的请求参数名称相同。

2、修改 ProductMapper.xml

拼接查询的 SQL 语句，查询结果总数和查询结果集都要拼接，且查询条件是一样的。

```
<select id="queryForCount" resultType="int">
    SELECT COUNT(*) FROM product
    <where>
        <if test="productName != null and productName != ''">
            AND productName LIKE concat('%', #{productName}, '%')
        </if>
        <if test="minSalePrice != null">
            AND salePrice &gt;= #{minSalePrice}
        </if>
        <if test="maxSalePrice != null">
            AND salePrice &lt;= #{maxSalePrice}
        </if>
    </where>
</select>

<select id="queryForList" resultType="cn.wolfcode.domain.Product">
    SELECT * FROM product
    <where>
        <if test="productName != null and productName != ''">
            AND productName LIKE concat('%', #{productName}, '%')
        </if>
        <if test="minSalePrice != null">
            AND salePrice &gt;= #{minSalePrice}
        </if>
        <if test="maxSalePrice != null">
            AND salePrice &lt;= #{maxSalePrice}
```

```
        </if>
    </where>
    LIMIT #{start}, #{pageSize}
</select>
```

3、修改单元测试类

造一些差异化的数据，修改 ProductServiceTest.java 其中 testQuery 方，测试过滤查询，看是否有问题，最好每个条件都试一下。

```
@Test
public void testQuery(){
    ProductQueryObject qo= new ProductQueryObject();
    qo.setCurrentPage(1);

    qo.setProductName("M");
    qo.setMinSalePrice(new BigDecimal("200"));
    qo.setMaxSalePrice(new BigDecimal("500"));
    PageResult<Product> pageResult = productService.query(qo);

    System.out.println("结果集数据：" + pageResult.getData());
    System.out.println("当前页总记录数：" + pageResult.getTotalCount());
    System.out.println("条数：" + pageResult.getData().size());
    System.out.println("总页数：" + pageResult.getTotalPage());
    System.out.println("上一页：" + pageResult.getPrevPage());
    System.out.println("下一页：" + pageResult.getNextPage());
}
```

4、修改 list.jsp

在 form 中增加过滤查询条件 input 元素，包含可以输入商品名，最小销售价，最大销售价，注意：让该表单（form）包含列表的 table 元素。

```
<form action="/product" method="post">
        货品名:<input type="text" name="productName" style="width: 80px;">
        价格: <input type="number" name="minSalePrice" style="width: 80px"> -
            <input type="number" name="maxSalePrice" style="width: 80px;">
            <input type="submit" value="查询">
        <table border="1" cellspacing="0" cellpadding="0" width="80%">
            <!-- 省略 -->
    </table>
</form>
```

5、修改 ProductServlet.java

在调用业务对象的 query 方法之前，获取过滤查询参数，并封装到 ProductQueryObject 对象中。

```
protected void list(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    ProductQueryObject qo = new ProductQueryObject();

    // 获取请求参数 productName
    String productName = req.getParameter("productName");
```

```
        qo.setProductName(productName);

        // 获取请求参数 minSalePrice
        String minSalePrice = req.getParameter("minSalePrice");
        if(StringUtil.hasLength(minSalePrice)){
            qo.setMinSalePrice(new BigDecimal(minSalePrice));
        }

        // 获取请求参数 maxSalePrice
        String maxSalePrice = req.getParameter("maxSalePrice");
        if(StringUtil.hasLength(maxSalePrice)){
            qo.setMaxSalePrice(new BigDecimal(maxSalePrice));
        }

        // 获取请求参数 currentPage
        String currentPage = req.getParameter("currentPage");
        if(StringUtil.hasLength(currentPage)) {
            qo.setCurrentPage(Integer.valueOf(currentPage));
        }

        // 获取请求参数 pageSize
        String pageSize = req.getParameter("pageSize");
        if(StringUtil.hasLength(pageSize)) {
            qo.setPageSize(Integer.valueOf(pageSize));
        }

        // 调用业务层方法来处理请求某一页数据的需求
        PageResult<Product> pageResult = productService.query(qo);
        // 把数据共享给 list.jsp
        req.setAttribute("pageResult", pageResult);
        // 控制跳转到 list.jsp 页面
        req.getRequestDispatcher("/WEB-INF/views/product/list.jsp").forward(req,
    resp);
    }
```


6、查询条件回显

修改 ProductServlet.java，在跳转 list.jsp 前加入如下代码：

```
// 把查询参数数据共享给 list.jsp
req.setAttribute("qo", qo);
```

修改 list.jsp，使之支持回显出查询的信息：

```
<form action="/product" method="post">
        货品名:<input type="text" name="productName" style="width: 80px;"
value="${qo.productName}">
        价格: <input type="number" name="minSalePrice" style="width: 80px"
value="${qo.minSalePrice}"> -
            <input type="number" name="maxSalePrice" style="width: 80px;"
value="${qo.maxSalePrice}">
            <input type="submit" value="查询">
        <table border="1" cellspacing="0" cellpadding="0" width="80%">
            <!-- 省略 -->
    </table>
</form>
```

7、页面测试

每个条件都要测试到，保证在网页上测试通过。

五、细节说明

1、concat 函数

MySQL 中 concat 函数是用来拼接字符串，项目模糊查询时会使用这个函数，与 % 进行拼接完成模糊查询的需求。

```
<!-- 其中 if 标签 和 #{} 中的 productName 指 ProductQueryObject 的属性名 -->
<!-- AND 后面的 productName 是 product 表中列明 -->
<if test="productName != null and productName != ''">
    AND productName LIKE concat('%', #{productName}, '%')
</if>
```

2、productName 判断空的问题

productName 属性是字符串类型，用户查询的时候，可能没传此参数或者传的是空字符串，所以我们需要在拼接条件之前进行判断，若用户查询传的是非空且有内容的的字符串时，才按照此条件过滤查询商品信息。

而判断方式可以有以下几种：

2.1、在 Mapper XML 中判断

```
<if test="productName != null and productName != ''">
    AND productName LIKE concat('%', #{productName}, '%')
</if>
```

2.2、在 Java 代码中判断

因为 MyBatis 通过调用查询对象中的 getProductName() 方法，获取到 productName 属性值，然后再做判断的。所以我们可以在把查询对象中的 getProductName() 方法重写，在 MyBatis 获取 productName 属性值之前加入判断。

```java
package cn.wolfcode.qo;

@Setter
@Getter
public class ProductQueryObject extends QueryObject {
    private String productName;
    private BigDecimal minSalePrice;
    private BigDecimal maxSalePrice;

    public String getProductName() {
        return StringUtil.hasLength(this.productName) ? this.productName : null;
    }
}
```

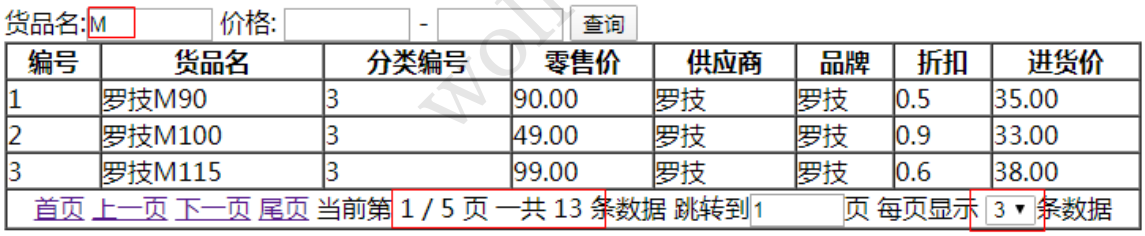
那么在 Mapper XML 中此时就不需要再做空字符串的判断了只须像下面这样写即可。

```xml
<if test="productName != null">
    AND productName LIKE concat('%', #{productName}, '%')
</if>
```

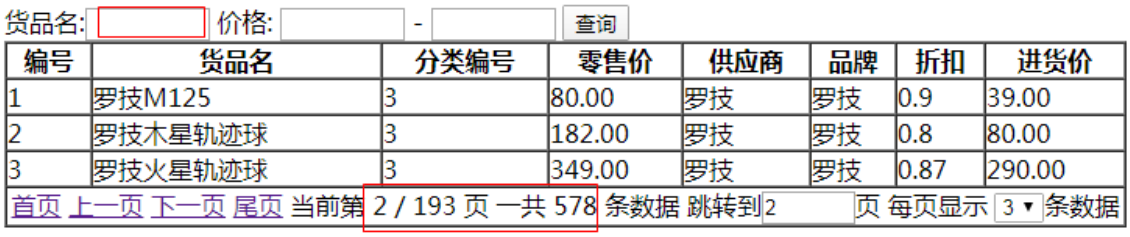
六、解决翻页数据丢失问题

1、存在的问题

当通过高级查询查询出结果集的时候，如下图：查询条件货品名输入的是 M，执行查询返回如下结果：



但之后如果再点击下一页：应该是进入 2/5 页，可是结果却是如下这样的：



此时，我们会发现过滤查询条件已经丢失，分页条的数据也全部和预期不一样了。

2、问题原因

此时的翻页操作（点击首页，上一页，下一页，尾页等这些操作），是通过 a 标签的超链接 发送 GET 请求查询，该超链接只带有 currentPage 参数，没带有其他过滤查询条件参数。

```
<a href="/product?currentPage=1">首页</a>
<a href="/product?currentPage=${pageResult.prevPage}">上一页</a>
<a href="/product?currentPage=${pageResult.nextPage}">下一页</a>
<a href="/product?currentPage=${pageResult.totalPage}">尾页</a>
```

3、解决思路

在翻页的时候，不仅要发送 currentPage 到后台，同时还需要把过滤查询的参数发送到后台。解决方案：

- 在超链接上拼接过滤查询的参数，若参数过多，实现比较麻烦。
- 使用 JS 来实现翻页操作，将超链接上的分页相关的参数放到表单中（推荐方式）。
  - 在高级查询表单中提供 currentPage 的文本框，我们把需要跳转的页码设置到该文本框。
  - 再使用 JS 提交过滤查询表单，此时就可以把查询条件和 currentPage 一起提交到后台，并封装到 ProductQueryObject中。

4、代码实现

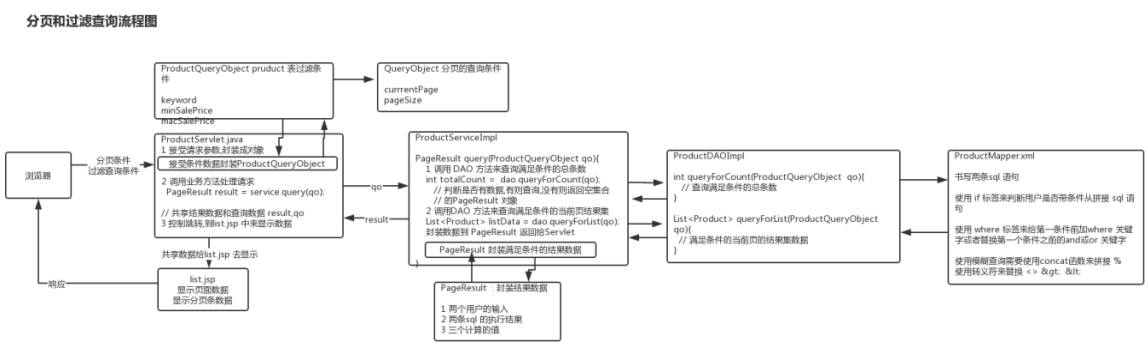
修改 list.jsp：

```
<!-- 加了 id 属性，目的为了方便通过 JS 获取标签对象重新设置 value 属性值 -->
跳转到<input type="number" id="currentPage" onchange="changePageSize()"
name="currentPage" value="${pageResult.currentPage}" style="width: 60px;">页
```

```
<a href="#" onclick="goPage(1);">首页</a>
<a href="#" onclick="goPage(${pageResult.prevPage});">上一页</a>
<a href="#" onclick="goPage(${pageResult.nextPage});">下一页</a>
<a href="#" onclick="goPage(${pageResult.totalPage});">尾页</a>
```

```
function goPage(paegNum) {
    // 把pageNum 设置给表单中表示跳转到某一页的表单控件中
    document.getElementById("currentPage").value = paegNum;
    // 提交表单数据
    document.forms[0].submit();
}
```

七、分页过滤查询流程图



八、拓展

1、Mapper XML 中代码优化

抽取重复 SQL 的片段（模板），供其他 SQL 包含。

```
<!-- 抽取一个 SQL 片段-->
<sql id="where_sql">
    <where>
        <if test="productName != null and productName != ''">
            AND productName LIKE concat('%', #{productName}, '%')
        </if>
        <if test="minSalePrice != null">
            AND salePrice &gt;= #{minSalePrice}
        </if>
        <if test="maxSalePrice != null">
            AND salePrice &lt;= #{maxSalePrice}
        </if>
    </where>
</sql>

<select id="queryForCount" resultType="int">
    SELECT COUNT(*) FROM product
    <include refid="where_sql"/>
</select>
<select id="queryForList" resultType="cn.wolfcode.domain.Product">
    SELECT * FROM product
    <include refid="where_sql"/>
    LIMIT #{start}, #{pageSize}
</select>
```


2、加入关键字查询

所谓关键字查询，根据输入的关键字，把表中多个列值包含这个关键字的查询出来。举个例子：有个需求根据产品的名字和品牌做关键字查询，其本质意思就是产品名称**或者**产品品牌只要包含输入的关键字就把其查询出来。