分页查询

一、分页查询

分页查询则是在页面上将本来很多的数据分段显示，每页显示用户自定义的行数。可提高用户体验度，同时减少一次性加载，内存溢出风险。

1、真假分页

分页分为：真分页和假分页。

- **假分页**：一次性查询所有数据存入内存，翻页从内存中获取数据。优点：实现简单，性能高；缺点：容易造成内存溢出。
- **真分页**：每次翻页从数据库中查询数据。优点：不容易造成内存溢出；缺点：实现复杂，性能相对低。

2、分页效果

发送请求访问一个带有分页页面 (京东) 的数据，会发现其主要由两部分组成：

- 当前页的**结果集**数据，比如这一页有哪些商品信息。
- **分页条信息**，比如包含【首页】【上页】【下页】【末页】等。

咱们的设计：

货品编号	货品名称	货品品牌	货品分类	供应商	零售价	成本价	折扣
1	罗技M90	罗技	有线鼠标	罗技	90.00	35.00	0.5
2	罗技M100	罗技	有线鼠标	罗技	49.00	33.00	0.9
3	罗技M115	罗技	有线鼠标	罗技	99.00	38.00	0.6
4	罗技M125	罗技	有线鼠标	罗技	80.00	39.00	0.9
5	罗技木星轨迹球	罗技	有线鼠标	罗技	182.00	80.00	0.8
6	罗技火星轨迹球	罗技	有线鼠标	罗技	460.68	290.00	0.87
7	罗技G9X	罗技	有线鼠标	罗技	897.60	470.00	0.7
8	罗技M215	罗技	无线鼠标	罗技	89.00	30.00	0.79
9	罗技M305	罗技	无线鼠标	罗技	119.00	48.00	0.82
10	罗技M310	罗技	无线鼠标	罗技	135.00	69.80	0.92
11	罗技M505	罗技	无线鼠标	罗技	148.00	72.00	0.92
12	罗技M555	罗技	无线鼠标	罗技	275.00	140.00	0.88
13	罗技M905	罗技	无线鼠标	罗技	549.60	270.00	0.88
14	罗技MX1100	罗技	无线鼠标	罗技	660.00	300.00	0.76
15	罗技M950	罗技	无线鼠标	罗技	813.60	320.00	0.78
首页 上页 下页 末页 当前第 3 / 7 页 一共 100条数据 跳转到[3]页 每页[15]条数据							

二、分页设计

从上面请求访问的分页演示的效果图分析分页是如何设计？

1、分页需传递的参数

需要用户传入的参数：

- currentPage：当前页，跳转到第几页，int 类型，设置默认值，比如 1。
- pageSize：每页最多多少条数据，int 类型，设置默认值，比如 10。

2、分页需展示的数据

从分页效果图中可以看出，分页需要依赖的数据：

- 当前页货品信息: data/list
- 分页条信息：
  - beginPage：　首页
  - prevPage：　上一页
  - nextPage：　下一页
  - totalPage：　总页数/末页
  - totalCount/rows：　总条数
  - currentPage：　当前页
  - pageSize：　每页显示多少条数据

3、分页需展示的数据的来源

- 来源于用户传入：
  - currentPage：　当前页，int 类型
  - pageSize：　每页显示多少条数据，int 类型
- 来源于两条 SQL 查询：
  - totalCount/rows：数据总条数，int 类型
  - data/list：　每一页的结果集数据，List 类型
- 来源于程序计算：
  - totalPage：　总页数/末页，int 类型
  - prevPage：　上一页，int 类型
  - nextPage：　下一页，int 类型

3.1、结果总数与结果集（分页原理）

结果总数（totalCount/rows）和结果集（data/list）是来源于两条 SQL（必须掌）的查询：

- **第一条 SQL**：查询符合条件的结果总数（totalCount/rows）

```
SELECT COUNT(*) FROM 表名 [WHERE 条件]
```

- **第二条 SQL**：查询符合条件的结果集（data/list）

```
# 第一个 ?：从哪一个索引的数据开始查询(默认从 0 开始)
# 第二个 ?：查询多少条数据
SELECT * FROM 表名 [WHERE 条件] LIMIT ?, ?
```

接下来分析第二条 SQL 中两个 ? 取值来源：

> 假设 product 表中有 21 条数据，每页分 5 条数据：
>
> 查询第一页数据：SELECT * FROM product LIMIT 0, 5
> 查询第二页数据：SELECT * FROM product LIMIT 5, 5
> 查询第三页数据：SELECT * FROM product LIMIT 10, 5
> 查询第四页数据：SELECT * FROM product LIMIT 15, 5
>
> 通过寻找规律发现：第一个 ? 取值来源于 (currentPage - 1) * pageSize；第二个 ? 取值来源于 pageSize，即都来源于用户传递的分页参数。

3.2、总页数、上一页和下一页

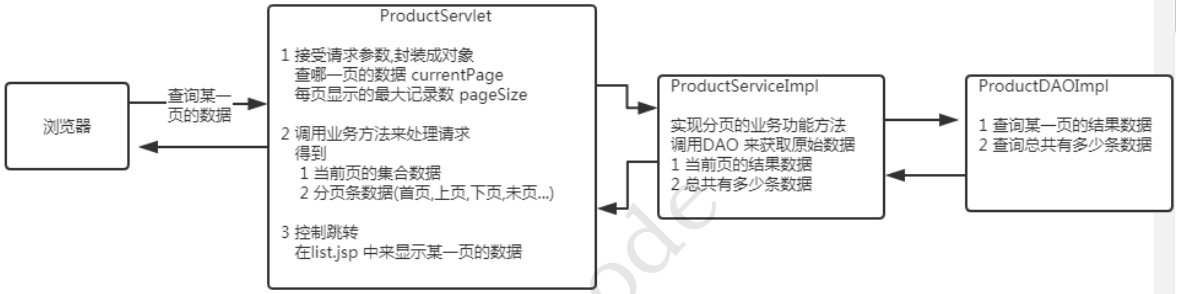
总页数、上一页和下一页都是来源于程序计算出来的。

```
int totalPage = rows % pageSize == 0 ? rows / pageSize : rows / pageSize  + 1;
// 优先计算
int prevPage = currentPage - 1 >= 1 ? currentPage - 1 : 1;
int nextPage = currentPage + 1 <= totalPage ? currentPage + 1 : totalPage;
```

三、分页查询实现

给产品增加分页查询的功能。

1、访问流程



2、分页数据封装

为了能在页面上显示上述的分页效果，那么我们就得在把页面上的每一个数据封装成到某个对象共享给JSP。

2.1、为什么要封装

若不封装的话，会怎样，效果如下：

Servlet代码：

```
把以下数据共享给JSP

request.setAttribute("products", list);
request.setAttribute("rows", 100);

request.setAttribute("prevPage", 2);
request.setAttribute("nextPage", 4);
request.setAttribute("totalPage", 7);

request.setAttribute("currentPage", 3);
request.setAttribute("pageSize", 15);
```

JSP代码：

```
当前第${currentPage} / ${rows} 页
```

恶心：数据太分散，需要共享多个数据，不方便统一管理多个数据。
解决方案：把多个需要共享的数据，封装到一个对象，往后就只需要把数据封装到该对象，再共享该对象即可。

2.2、编写 PageResult.java

在之前 Web CRUD 项目的基础上增加新的代码即可。

```
package cn.wolfcode.qo;

/**
 * 封装结果数据（某一页的数据）
 */
@Getter
public class PageResult<T> {
    // 两个用户的输入
    private int currentPage;        // 当前页码
    private int pageSize;           // 每页显示的条数
    // 两条 SQL 语句执行的结果
    private int totalCount;         // 总条数
    private List<T> data;           // 当前页结果集数据
    // 三个程序计算的数据
    private int prevPage;           // 上一页页码
    private int nextPage;           // 下一页页码
    private int totalPage;          // 总页数/末页页码

    // 分页数据通过下面构造期封装好，但思考一下这个构造器什么调用？
    public PageResult(int currentPage, int pageSize, int totalCount, List<T>
data) {
        this.currentPage = currentPage;
        this.pageSize = pageSize;
        this.totalCount = totalCount;
        this.data = data;

        // 计算三个数据
        this.totalPage = totalCount % pageSize == 0 ? totalCount / pageSize :
totalCount / pageSize + 1;
        this.prevPage = currentPage - 1 >= 1 ? currentPage - 1 : 1;
        this.nextPage = currentPage + 1 <= this.totalPage ? currentPage + 1 :
this.totalPage;
    }
}
```

3、持久层分页功能实现

分页其最终的所有功能需要依赖的 SQL 就两条：

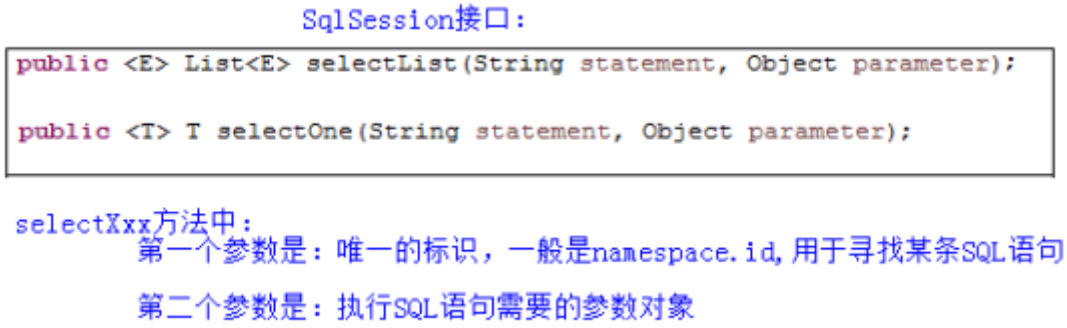
- 查询数据的总数（为了显示分页条信息）。
- 查询当前页的数据。
  所以我们就得给 DAO 增加两个方法，一个查询数据总量，一个查询当前页的数据。

3.1、修改 IProductDAO.java

给其增加两个方法，分别用于查询结果总数和结果集。

```
int queryForCount(int currentPage, int pageSize);
List<Product> queryForList(int currentPage, int pageSize);
```

上述这样写的话方法是有两个形参的，但是 MyBatis 提供的操作方法传入执行 SQL 任务的参数，**注意只能是一个**，而现在两个参数需要传递两个分页的实参。



解决方案:
方案一：使用 Map 来封装需要传递的参数。
方案二：使用 JavaBean 来封装需要传递的参数。

3.2、编写 QueryObject.java

目前用来封装分页参数，并解决上面的问题。

```
package cn.wolfcode.qo;

@Setter
@Getter
/**
 * 封装分页查询需要的两个请求传入的分页参数
 */
public class QueryObject {
    private int currentPage = 1;    // 当前页码，要跳转到哪一页的页码（需要给默认值）
    private int pageSize = 3;       // 每页显示条数（需要给默认值）
}
```


3.3、再修改 IProductDAO.java

修改分页查询的两个方法，改方法形参为 QueryObject。

```
int queryForCount(QueryObject qo);
List<Product> queryForList(QueryObject qo);
```

3.4、修改 ProductDAOImpl.java

```
@Override
public int queryForCount(QueryObject qo) {
    SqlSession session = MyBatisUtil.getSession();
    int totalCount =
session.selectOne("cn.wolfcode.mapper.ProductMapper.queryForCount", qo);
    session.close();
    return totalCount;
}

@Override
public List<Product> queryForList(QueryObject qo) {
    SqlSession session = MyBatisUtil.getSession();
```

```
    List<Product> products =
session.selectList("cn.wolfcode.mapper.ProductMapper.queryForList",qo);
    session.close();
    return products;
}
```

3.5、修改 ProductMapper.xml

```
<select id="queryForCount" resultType="int">
    SELECT COUNT(*) FROM product
</select>

<select id="queryForList" resultType="cn.wolfcode.domain.Product">
    SELECT * FROM product LIMIT #{start}, #{pageSize}
</select>
```

3.6、修改 QueryObjet.java

给 QueryObject 增加 getStart 方法，返回之前找出规律算出从那个位置开始查询数据，代码如下：

```
package cn.wolfcode.qo;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
/**
 * 封装分页查询需要的两个请求传入的分页参数
 */
public class QueryObject {
    private int currentPage = 1;     // 当前页码，要跳转到哪一页的页码（需要给默认值）
    private int pageSize = 3;        // 每页显示条数（需要给默认值）

    // 用于 Limit 子句第一个 ? 取值
    public int getStart(){
        return (currentPage - 1) * pageSize;
    }
}
```


3.7、编写单元测试

给测试类 ProductDAOTest 中添加分页测试的方法，测试一下。

```
 @Test
public void testQueryForCount(){
    QueryObject qo = new QueryObject();
    System.out.println(productDAO.queryForCount(qo));
}

@Test
public void testQueryForList(){
    QueryObject qo= new QueryObject();
    System.out.println(productDAO.queryForList(qo));
}
```

4、业务层分页功能实现

给 Servlet 调用，返回分页查询的结果，即返回 PageResult 对象，而形参类型为 QueryObject。

4.1、编写 IProductService.java

```
package cn.wolfcode.service;

public interface IProductService {
    /**
     * 完成查询某一页的业务逻辑功能
     */
    PageResult<Product> query(QueryObject qo);
}
```


4.2、编写 ProductServiceImpl.java

```
package cn.wolfcode.service.impl;

public class ProductServiceImpl implements IProductService {

    private IProductDAO productDAO = new ProductDAOImpl();

    @Override
    public PageResult<Product> query(QueryObject qo) {
        // 调用 DAO 查询数据数量
        int totalCount = productDAO.queryForCount(qo);
        // 为了性能加入判断，若查询的数据数量为 0，说明没有数据，返回返回空集合，即集合中没有
元素
        if(totalCount == 0){
            return new PageResult(qo.getCurrentPage(), qo.getPageSize(),
totalCount, Collections.emptyList());
        }
        // 执行到这里代表有数据，查询当前页的结果数据
        List<Product> products = productDAO.queryForList(qo);
        return new PageResult(qo.getCurrentPage(), qo.getPageSize(), totalCount,
products);
    }
}
```

4.3、编写单元测试类

在 test 目录下的 cn.wolfcode.service 包下新建 ProductServiceTest 测试类，测试上面 query 方法，观察控制台输出结果。

```
package cn.wolfcode.service;

public class ProductServiceTest {

    private IProductService productService = new ProductServiceImpl();

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

5、前台分页功能实现

包含编写 Servlet 及 JSP，Servlet 处理请求，调用业务方法，把查询到数据共享到 JSP 中，展示给用户看。

5.1、操作步骤

- **必须先完成业务层组件，保证后台测试通过。**
- 遵循 MVC 思想。
- 浏览器发出分页请求参数（去往第几页/每页多少条数据），在 Servlet 中接收这些参数，并封装到 QueryObject 对象，调用 Service 中分页查询方法（query）。
- 把得到的分页查询结果对象（PageResult）共享在请求作用域中，跳转到 JSP，显示即可。
- 修改 JSP 页面，编写出分页条信息（分页条中的信息来源于 PageResult 对象）。

5.2、修改 ProductServlet.java

ProductServlet，获取页面传递的分页参数，执行查询，将结果共享到请求作用域，请求转发回到 list.jsp 页面。

注意暂时注释掉其他报错代码，因为目前产品业务类中只提供分页查询的方法。

```
private IProductService productService = new ProductServiceImpl();

protected void list(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    QueryObject qo = new QueryObject();
    // 获取请求参数 currentPage，并转型封装
    String currentPage = req.getParameter("currentPage");
    if(StringUtil.hasLength(currentPage)) {
        qo.setCurrentPage(Integer.valueOf(currentPage));
    }
```

```
    // 获取请求参数 pageSize，并转型封装
    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查询某一页数据
    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);
}
```

5.3、修改 list.jsp

在此 JSP 使用 JSTL + EL 获取后台共享到请求作用域中的数据，展示给用户看。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>产品列表</title>
    <script type="text/javascript">
        window.onload = function () {
            var trClzs = document.getElementsByClassName("trClassName");
            for(var i = 0; i < trClzs.length; i++){
                trClzs[i].onmouseover = function () {
                    console.log(1);
                    this.style.backgroundColor = "gray";
                }
                trClzs[i].onmouseout = function () {
                    console.log(2);
                    this.style.backgroundColor = "";
                }
            }
        }

        // 分页 JS
        function changePageSize() {
            document.forms[0].submit();
        }
    </script>
</head>
<body>
    <a href="/replaceImg.jsp"><img src="${USER_IN_SESSION.headImg}" title="更换头
像"/></a><br/>
    <a href="/product?cmd=input">添加</a>
    <form action="/product">
        <table border="1" cellspacing="0" cellpadding="0" width="80%">
            <tr>
                <th>编号</th>
                <th>货品名</th>
                <th>分类编号</th>
                <th>零售价</th>
                <th>供应商</th>
```

```
                <th>品牌</th>
                <th>折扣</th>
                <th>进货价</th>
                <th>操作</th>
            </tr>
            <c:forEach var="product" items="${pageResult.data}"
varStatus="status">
                <tr class="trClassName">
                    <td>${status.count}</td>
                    <td>${product.productName}</td>
                    <td>${product.dir_id}</td>
                    <td>${product.salePrice}</td>
                    <td>${product.supplier}</td>
                    <td>${product.brand}</td>
                    <td>${product.cutoff}</td>
                    <td>${product.costPrice}</td>
                    <td>
                        <a href="/product?cmd=delete&id=${product.id}">删除</a>
                        <a href="/product?cmd=input&id=${product.id}">修改</a>
                    </td>
                </tr>
            </c:forEach>
            <tr align="center">
                <td colspan="9">
                    <a href="/product?currentPage=1">首页</a>
                    <a href="/product?currentPage=${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>
                    当前第 ${pageResult.currentPage} / ${pageResult.totalPage} 页
                    一共 ${pageResult.totalCount} 条数据
                    跳转到<input type="number" onchange="changePageSize()"
name="currentPage" value="${pageResult.currentPage}" style="width: 60px;">页
                    每页显示
                    <select name="pageSize" onchange="changePageSize()">
                        <option value="3" ${pageResult.pageSize == 3 ?
'selected' : ''}> 3 </option>
                        <option value="5" ${pageResult.pageSize == 5 ?
'selected' : ''}> 5 </option>
                        <option value="8" ${pageResult.pageSize == 8 ?
'selected' : ''}> 8 </option>
                    </select>条数据
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

四、常见问题

疑惑理解:

- currentPage，表示要显示某一页数据的页码，在程序中表示翻页要去往的页面，比如 curentPage = 3，表示要跳转到第 3 页。
- 上一页，下一页，末页是在 PageResult 的构造器中已经计算好。

常见问题：若刚开始翻页操作成功，但是翻几页之后就不能翻页了，只能重启 Tomcat 才可以翻页，但是操作几次又不能翻页了。问题原因：在 DAO 中没有关闭 SqlSession 对象释放资源。

Servlet 代码优化：

```
protected void list(...) {
    req2qo(req, qo);
    // ......
}

// 获取请求参数封装成对象
private void req2qo(HttpServletRequest req, QueryObject qo) {
    String currentPage = req.getParameter("currentPage");
    String pageSize = req.getParameter("pageSize");

    if(StringUtil.hasLength(currentPage)) {
        qo.setCurrentPage(Integer.valueOf(currentPage));
    }

    if(StringUtil.hasLength(pageSize)) {
        qo.setPageSize(Integer.valueOf(pageSize));
    }
}
```