自：https://www.cnblogs.com/digdeep/p/4608933.html

[**Mybatis 数据库物理分页插件 PageHelper**](https://www.cnblogs.com/digdeep/p/4608933.html)

以前使用ibatis/mybatis，都是自己手写sql语句进行物理分页，虽然稍微有点麻烦，但是都习惯了。最近试用了下mybatis的分页插件 PageHelper,感觉还不错吧。记录下其使用方法。

**1.** 引入依赖jar包：

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

<version>3.7.5</version>

</dependency>

**2.** 配置分页拦截器

PageHelper的原理是基于拦截器实现的。拦截器的配置有两种方法，一种是在mybatis的配置文件中配置，一种是直接在spring的配置文件中进行：

1）在mybatis-config.xml文件中配置：

[复制代码](javascript:void(0);)

<plugins>

<!-- com.github.pagehelper为PageHelper类所在包名 -->

<plugin interceptor="com.github.pagehelper.PageHelper">

<property name="dialect" value="mysql"/>

<!-- 该参数默认为false -->

<!-- 设置为true时，会将RowBounds第一个参数offset当成pageNum页码使用 -->

<!-- 和startPage中的pageNum效果一样-->

<property name="offsetAsPageNum" value="true"/>

<!-- 该参数默认为false -->

<!-- 设置为true时，使用RowBounds分页会进行count查询 -->

<property name="rowBoundsWithCount" value="true"/>

<!-- 设置为true时，如果pageSize=0或者RowBounds.limit = 0就会查询出全部的结果 -->

<!-- （相当于没有执行分页查询，但是返回结果仍然是Page类型）

<property name="pageSizeZero" value="true"/>-->

<!-- 3.3.0版本可用 - 分页参数合理化，默认false禁用 -->

<!-- 启用合理化时，如果pageNum<1会查询第一页，如果pageNum>pages会查询最后一页 -->

<!-- 禁用合理化时，如果pageNum<1或pageNum>pages会返回空数据 -->

<property name="reasonable" value="true"/>

<!-- 3.5.0版本可用 - 为了支持startPage(Object params)方法 -->

<!-- 增加了一个`params`参数来配置参数映射，用于从Map或ServletRequest中取值 -->

<!-- 可以配置pageNum,pageSize,count,pageSizeZero,reasonable,不配置映射的用默认值 -->

<!-- 不理解该含义的前提下，不要随便复制该配置

<property name="params" value="pageNum=start;pageSize=limit;"/> -->

</plugin>

</plugins>

[复制代码](javascript:void(0);)

这里要注意 <plugins> 在mybatis-config.xml文件中的位置，必须要符合 http://mybatis.org/dtd/mybatis-3-config.dtd 中指定的顺序：

<!ELEMENT configuration (properties?, settings?, typeAliases?, typeHandlers?,   
 objectFactory?, objectWrapperFactory?, plugins?, environments?, databaseIdProvider?, mappers?)>

不然会报错。

当然mybatis-config.xml的位置，我们需要在spring的配置文件中进行指定：

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="configLocation" value="classpath:config/mybatis-config.xml" />

<property name="mapperLocations" value="classpath\*:config/mappers/\*\*/\*.xml" />

</bean>

2）如果mybatis没有mybatis-config.xml文件，那么就只能直接在spring的配置文件中配置了：

[复制代码](javascript:void(0);)

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="mapperLocations">

<array>

<value>classpath:config/mapper/\*.xml</value>

</array>

</property>

<property name="typeAliasesPackage" value="com.test.pojo"/>

<property name="plugins">

<array>

<bean class="com.github.pagehelper.PageHelper">

<property name="properties">

<value>

dialect=mysql

</value>

</property>

</bean>

</array>

</property>

</bean>

[复制代码](javascript:void(0);)

到这里PageHelper所需要的配置已经完成，下面还需要在serviceImpl类中加入分页参数的代码：

**3.** 向拦截器传递分页参数

我们首先看下不分页的serviceImpl代码：

[复制代码](javascript:void(0);)

@Override

public List<User> getUserByNoAndEmail(String no, String email) {

Map<String, Object> map = new HashMap<>();

map.put("no", no);

map.put("email", email);

return this.userMapper.getUserByNoAndEmail(map);

}

[复制代码](javascript:void(0);)

然后我们将它改造成使用PageHelper分页：

1）首先我们根据自己项目的情况，定义一个PageBean，来保存分页之后的结果，需要哪些属性，就加入哪些属性，具体可以参考源代码中的PageInfo类的定义，其实**PageInfo是插件作者给我们自己定义自己的PageBean，提供的一个参考例子**。PageInfo代码如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

@SuppressWarnings({"rawtypes", "unchecked"})

public class PageInfo<T> implements Serializable {

private static final long serialVersionUID = 1L;

//当前页

private int pageNum;

//每页的数量

private int pageSize;

//当前页的数量

private int size;

//由于startRow和endRow不常用，这里说个具体的用法

//可以在页面中"显示startRow到endRow 共size条数据"

//当前页面第一个元素在数据库中的行号

private int startRow;

//当前页面最后一个元素在数据库中的行号

private int endRow;

//总记录数

private long total;

//总页数

private int pages;

//结果集

private List<T> list;

//第一页

private int firstPage;

//前一页

private int prePage;

//下一页

private int nextPage;

//最后一页

private int lastPage;

//是否为第一页

private boolean isFirstPage = false;

//是否为最后一页

private boolean isLastPage = false;

//是否有前一页

private boolean hasPreviousPage = false;

//是否有下一页

private boolean hasNextPage = false;

//导航页码数

private int navigatePages;

//所有导航页号

private int[] navigatepageNums;

/\*\*

\* 包装Page对象

\*

\* @param list

\*/

public PageInfo(List<T> list) {

this(list, 8);

}

/\*\*

\* 包装Page对象

\*

\* @param list page结果

\* @param navigatePages 页码数量

\*/

public PageInfo(List<T> list, int navigatePages) {

if (list instanceof Page) {

Page page = (Page) list;

this.pageNum = page.getPageNum();

this.pageSize = page.getPageSize();

this.total = page.getTotal();

this.pages = page.getPages();

this.list = page;

this.size = page.size();

//由于结果是>startRow的，所以实际的需要+1

if (this.size == 0) {

this.startRow = 0;

this.endRow = 0;

} else {

this.startRow = page.getStartRow() + 1;

//计算实际的endRow（最后一页的时候特殊）

this.endRow = this.startRow - 1 + this.size;

}

this.navigatePages = navigatePages;

//计算导航页

calcNavigatepageNums();

//计算前后页，第一页，最后一页

calcPage();

//判断页面边界

judgePageBoudary();

}

}

/\*\*

\* 计算导航页

\*/

private void calcNavigatepageNums() {

//当总页数小于或等于导航页码数时

if (pages <= navigatePages) {

navigatepageNums = new int[pages];

for (int i = 0; i < pages; i++) {

navigatepageNums[i] = i + 1;

}

} else { //当总页数大于导航页码数时

navigatepageNums = new int[navigatePages];

int startNum = pageNum - navigatePages / 2;

int endNum = pageNum + navigatePages / 2;

if (startNum < 1) {

startNum = 1;

//(最前navigatePages页

for (int i = 0; i < navigatePages; i++) {

navigatepageNums[i] = startNum++;

}

} else if (endNum > pages) {

endNum = pages;

//最后navigatePages页

for (int i = navigatePages - 1; i >= 0; i--) {

navigatepageNums[i] = endNum--;

}

} else {

//所有中间页

for (int i = 0; i < navigatePages; i++) {

navigatepageNums[i] = startNum++;

}

}

}

}

/\*\*

\* 计算前后页，第一页，最后一页

\*/

private void calcPage() {

if (navigatepageNums != null && navigatepageNums.length > 0) {

firstPage = navigatepageNums[0];

lastPage = navigatepageNums[navigatepageNums.length - 1];

if (pageNum > 1) {

prePage = pageNum - 1;

}

if (pageNum < pages) {

nextPage = pageNum + 1;

}

}

}

/\*\*

\* 判定页面边界

\*/

private void judgePageBoudary() {

isFirstPage = pageNum == 1;

isLastPage = pageNum == pages;

hasPreviousPage = pageNum > 1;

hasNextPage = pageNum < pages;

}

public void setPageNum(int pageNum) {

this.pageNum = pageNum;

}

public int getPageNum() {

return pageNum;

}

public int getPageSize() {

return pageSize;

}

public int getSize() {

return size;

}

public int getStartRow() {

return startRow;

}

public int getEndRow() {

return endRow;

}

public long getTotal() {

return total;

}

public int getPages() {

return pages;

}

public List<T> getList() {

return list;

}

public int getFirstPage() {

return firstPage;

}

public int getPrePage() {

return prePage;

}

public int getNextPage() {

return nextPage;

}

public int getLastPage() {

return lastPage;

}

public boolean isIsFirstPage() {

return isFirstPage;

}

public boolean isIsLastPage() {

return isLastPage;

}

public boolean isHasPreviousPage() {

return hasPreviousPage;

}

public boolean isHasNextPage() {

return hasNextPage;

}

public int getNavigatePages() {

return navigatePages;

}

public int[] getNavigatepageNums() {

return navigatepageNums;

}

@Override

public String toString() {

final StringBuffer sb = new StringBuffer("PageInfo{");

sb.append("pageNum=").append(pageNum);

sb.append(", pageSize=").append(pageSize);

sb.append(", size=").append(size);

sb.append(", startRow=").append(startRow);

sb.append(", endRow=").append(endRow);

sb.append(", total=").append(total);

sb.append(", pages=").append(pages);

sb.append(", list=").append(list);

sb.append(", firstPage=").append(firstPage);

sb.append(", prePage=").append(prePage);

sb.append(", nextPage=").append(nextPage);

sb.append(", lastPage=").append(lastPage);

sb.append(", isFirstPage=").append(isFirstPage);

sb.append(", isLastPage=").append(isLastPage);

sb.append(", hasPreviousPage=").append(hasPreviousPage);

sb.append(", hasNextPage=").append(hasNextPage);

sb.append(", navigatePages=").append(navigatePages);

sb.append(", navigatepageNums=");

if (navigatepageNums == null) sb.append("null");

else {

sb.append('[');

for (int i = 0; i < navigatepageNums.length; ++i)

sb.append(i == 0 ? "" : ", ").append(navigatepageNums[i]);

sb.append(']');

}

sb.append('}');

return sb.toString();

}

}

[复制代码](javascript:void(0);)

因为PageInfo.java只是一个示例，所以他定义得有点重量级，属性有点多，我们可以参考它，定义适合我们自己的PageBean, 比如如下定义：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

public class PageBean<T> implements Serializable {

private static final long serialVersionUID = 8656597559014685635L;

private long total; //总记录数

private List<T> list; //结果集

private int pageNum; // 第几页

private int pageSize; // 每页记录数

private int pages; // 总页数

private int size; // 当前页的数量 <= pageSize，该属性来自ArrayList的size属性

/\*\*

\* 包装Page对象，因为直接返回Page对象，在JSON处理以及其他情况下会被当成List来处理，

\* 而出现一些问题。

\* @param list page结果

\* @param navigatePages 页码数量

\*/

public PageBean(List<T> list) {

if (list instanceof Page) {

Page<T> page = (Page<T>) list;

this.pageNum = page.getPageNum();

this.pageSize = page.getPageSize();

this.total = page.getTotal();

this.pages = page.getPages();

this.list = page;

this.size = page.size();

}

}

public long getTotal() {

return total;

}

public void setTotal(long total) {

this.total = total;

}

public List<T> getList() {

return list;

}

public void setList(List<T> list) {

this.list = list;

}

public int getPageNum() {

return pageNum;

}

public void setPageNum(int pageNum) {

this.pageNum = pageNum;

}

public int getPageSize() {

return pageSize;

}

public void setPageSize(int pageSize) {

this.pageSize = pageSize;

}

public int getPages() {

return pages;

}

public void setPages(int pages) {

this.pages = pages;

}

public int getSize() {

return size;

}

public void setSize(int size) {

this.size = size;

}

}

[复制代码](javascript:void(0);)

**因为分页查询结果返回的是一个 Page 对象，而 Page 对象继承自ArrayList，但是如果我们直接返回ArrayList的话，在一些场景下回遇到问题，比如在JSON处理Page类型的结果时，会被当成List来JSON格式化，会丢弃 Page 对象的所有扩展属性**，所以这里我们要**将分页的结果 Page 类型转换成我们自己定义的 PageBean. 我们自己定义的PageBean没有继承ArrayList，而是包含一个List属性来保存分页结果**。所以避免前面的问题。

2）修改 serviceImpl中的代码：

[复制代码](javascript:void(0);)

@Override

public PageBean<User> getUserByNoAndEmail(String no, String email) {

Map<String, Object> map = new HashMap<>();

map.put("no", no);

map.put("email", email);

PageHelper.startPage(PaginationContext.getPageNum(), PaginationContext.getPageSize());

List<User> list = this.userMapper.getUserByNoAndEmail(map);

return new PageBean<User>(list);

}

[复制代码](javascript:void(0);)

我们只需要使用 PageHelper.startPage(pageNum, pageSize); 函数来指定 pageNum(第几页) 和 pageSize(每页显示几条记录) 两个参数。然后调用原来的查询，就进行了分页。最后将返回的List，转换成 PageBean类型的结果即可。前台页面就可以根据PageBean中包括的属性来进行分页显示了。

上面的 PaginationContext 是基于 ThreadLocal 来传递分页参数的一个工具类，其实现如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

public class PaginationContext {

// 定义两个threadLocal变量：pageNum和pageSize

private static ThreadLocal<Integer> pageNum = new ThreadLocal<Integer>(); // 保存第几页

private static ThreadLocal<Integer> pageSize = new ThreadLocal<Integer>(); // 保存每页记录条数

/\*

\* pageNum ：get、set、remove

\*/

public static int getPageNum() {

Integer pn = pageNum.get();

if (pn == null) {

return 0;

}

return pn;

}

public static void setPageNum(int pageNumValue) {

pageNum.set(pageNumValue);

}

public static void removePageNum() {

pageNum.remove();

}

/\*

\* pageSize ：get、set、remove

\*/

public static int getPageSize() {

Integer ps = pageSize.get();

if (ps == null) {

return 0;

}

return ps;

}

public static void setPageSize(int pageSizeValue) {

pageSize.set(pageSizeValue);

}

public static void removePageSize() {

pageSize.remove();

}

}

[复制代码](javascript:void(0);)

实现了前台页面向ServiceImpl中传递分页参数： pageNum 和 pageSize.

当然从请求中获取分页参数pageNum和pageSize需要用到filter:

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

public class PageFilter implements Filter {

public PageFilter() {}

public void destroy() {}

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

HttpServletRequest httpRequest = (HttpServletRequest) request;

PaginationContext.setPageNum(getPageNum(httpRequest));

PaginationContext.setPageSize(getPageSize(httpRequest));

try {

chain.doFilter(request, response);

}

// 使用完Threadlocal，将其删除。使用finally确保一定将其删除

finally {

PaginationContext.removePageNum();

PaginationContext.removePageSize();

}

}

/\*\*

\* 获得pager.offset参数的值

\*

\* @param request

\* @return

\*/

protected int getPageNum(HttpServletRequest request) {

int pageNum = 1;

try {

String pageNums = request.getParameter("pageNum");

if (pageNums != null && StringUtils.isNumeric(pageNums)) {

pageNum = Integer.parseInt(pageNums);

}

} catch (NumberFormatException e) {

e.printStackTrace();

}

return pageNum;

}

/\*\*

\* 设置默认每页大小

\*

\* @return

\*/

protected int getPageSize(HttpServletRequest request) {

int pageSize = 10; // 默认每页10条记录

try {

String pageSizes = request.getParameter("pageSize");

if (pageSizes != null && StringUtils.isNumeric(pageSizes)) {

pageSize = Integer.parseInt(pageSizes);

}

} catch (NumberFormatException e) {

e.printStackTrace();

}

return pageSize;

}

/\*\*

\* @see Filter#init(FilterConfig)

\*/

public void init(FilterConfig fConfig) throws ServletException {}

}

[复制代码](javascript:void(0);)

PageFilter在web.xml中的配置：

[复制代码](javascript:void(0);)

<!-- pagination filter -->

<filter>

<filter-name>PageFilter</filter-name>

<filter-class>com.ems.filter.PageFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>PageFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

[复制代码](javascript:void(0);)

OK，到此，PageHelper的使用方法，基本结束。

PageHelper 项目地址：http://git.oschina.net/free/Mybatis\_PageHelper

文档地址：http://git.oschina.net/free/Mybatis\_PageHelper/blob/master/wikis/HowToUse.markdown

分类: [Java-ibatis/mybatis](https://www.cnblogs.com/digdeep/category/644918.html),[Java-spring/springmvc](https://www.cnblogs.com/digdeep/category/683618.html)

[«](https://www.cnblogs.com/digdeep/p/4607131.html)上一篇：[require.js 最佳实践](https://www.cnblogs.com/digdeep/p/4607131.html)  
[»](https://www.cnblogs.com/digdeep/p/4610844.html)下一篇：[SpringMVC 的 Controller 返回各种视图的处理方式](https://www.cnblogs.com/digdeep/p/4610844.html)

posted on 2015-06-29 22:15 [digdeep](https://www.cnblogs.com/digdeep/) 阅读(71263) 评论(2) [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=4608933) [收藏](https://www.cnblogs.com/digdeep/p/4608933.html)

**评论**

[**#1楼**](https://www.cnblogs.com/digdeep/p/4608933.html#3732981)**2017-07-11 17:47**[**lizhenlzlz**](https://www.cnblogs.com/lizhenlzlz/)

对我有用，非常好

[支持(0)反对(0)](javascript:void(0);)

[**#2楼**](https://www.cnblogs.com/digdeep/p/4608933.html#3817161)**2017-10-20 23:18**[**final变量**](http://home.cnblogs.com/u/1242291/)

Mybatis 数据库物理分页插件PageHelper - digdeep - 博客园，写的不错不错，收藏了。  
  
推荐下，分库分表中间件 Sharding-JDBC 源码解析 17 篇：<http://www.yunai.me/categories/Sharding-JDBC/?cnblog>&602