**使用方法**

**1. 引入分页插件**

引入分页插件有下面2种方式，推荐使用 Maven 方式。

**1). 引入 Jar 包**

你可以从下面的地址中下载最新版本的 jar 包

* <https://oss.sonatype.org/content/repositories/releases/com/github/pagehelper/pagehelper/>
* <http://repo1.maven.org/maven2/com/github/pagehelper/pagehelper/>

由于使用了sql 解析工具，你还需要下载 jsqlparser.jar：

* <http://repo1.maven.org/maven2/com/github/jsqlparser/jsqlparser/0.9.5/>

2). 使用 Maven

在 pom.xml 中添加如下依赖：

1. <dependency>
2. <groupId>com.github.pagehelper</groupId>
3. <artifactId>pagehelper</artifactId>
4. <version>最新版本</version>
5. </dependency>

最新版本号可以从首页查看。

**2. 配置拦截器插件**

特别注意，新版拦截器是 com.github.pagehelper.PageInterceptor。 com.github.pagehelper.PageHelper 现在是一个特殊的 dialect 实现类，是分页插件的默认实现类，提供了和以前相同的用法。

1. 在 MyBatis 配置 xml 中配置拦截器插件

1. *<!--*
2. *plugins在配置文件中的位置必须符合要求，否则会报错，顺序如下:*
3. *properties?, settings?,*
4. *typeAliases?, typeHandlers?,*
5. *objectFactory?,objectWrapperFactory?,*
6. *plugins?,*
7. *environments?, databaseIdProvider?, mappers?*
8. *-->*
9. <plugins>
10. *<!-- com.github.pagehelper为PageHelper类所在包名 -->*
11. <plugin interceptor="com.github.pagehelper.PageInterceptor">
12. *<!-- 使用下面的方式配置参数，后面会有所有的参数介绍 -->*
13. <property name="param1" value="value1"/>
14. </plugin>
15. </plugins>

**2.**在 Spring 配置文件中配置拦截器插件

使用 spring 的属性配置方式，可以使用 plugins 属性像下面这样配置：

1. <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
2. *<!-- 注意其他配置 -->*
3. <property name="plugins">
4. <array>
5. <bean class="com.github.pagehelper.PageInterceptor">
6. <property name="properties">
7. *<!--使用下面的方式配置参数，一行配置一个 -->*
8. <value>
9. params=value1
10. </value>
11. </property>
12. </bean>
13. </array>
14. </property>
15. </bean>

**3. 分页插件参数介绍**

分页插件提供了多个可选参数，这些参数使用时，按照上面两种配置方式中的示例配置即可。

分页插件可选参数如下：

* dialect：默认情况下会使用 PageHelper 方式进行分页，如果想要实现自己的分页逻辑，可以实现 Dialect(com.github.pagehelper.Dialect) 接口，然后配置该属性为实现类的全限定名称。

下面几个参数都是针对默认 dialect 情况下的参数。使用自定义 dialect 实现时，下面的参数没有任何作用。

1. helperDialect：分页插件会自动检测当前的数据库链接，自动选择合适的分页方式。 你可以配置helperDialect属性来指定分页插件使用哪种方言。配置时，可以使用下面的缩写值：  
   oracle,mysql,mariadb,sqlite,hsqldb,postgresql,db2,sqlserver,informix,h2,sqlserver2012,derby  
   特别注意：使用 SqlServer2012 数据库时，需要手动指定为 sqlserver2012，否则会使用 SqlServer2005 的方式进行分页。  
   你也可以实现 AbstractHelperDialect，然后配置该属性为实现类的全限定名称即可使用自定义的实现方法。
2. offsetAsPageNum：默认值为 false，该参数对使用 RowBounds 作为分页参数时有效。 当该参数设置为 true 时，会将 RowBounds 中的 offset 参数当成 pageNum 使用，可以用页码和页面大小两个参数进行分页。
3. rowBoundsWithCount：默认值为false，该参数对使用 RowBounds 作为分页参数时有效。 当该参数设置为true时，使用 RowBounds 分页会进行 count 查询。
4. pageSizeZero：默认值为 false，当该参数设置为 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）。
5. reasonable：分页合理化参数，默认值为false。当该参数设置为 true 时，pageNum<=0 时会查询第一页， pageNum>pages（超过总数时），会查询最后一页。默认false 时，直接根据参数进行查询。
6. params：为了支持startPage(Object params)方法，增加了该参数来配置参数映射，用于从对象中根据属性名取值， 可以配置 pageNum,pageSize,count,pageSizeZero,reasonable，不配置映射的用默认值， 默认值为pageNum=pageNum;pageSize=pageSize;count=countSql;reasonable=reasonable;pageSizeZero=pageSizeZero。
7. supportMethodsArguments：支持通过 Mapper 接口参数来传递分页参数，默认值false，分页插件会从查询方法的参数值中，自动根据上面 params 配置的字段中取值，查找到合适的值时就会自动分页。 使用方法可以参考测试代码中的 com.github.pagehelper.test.basic 包下的 ArgumentsMapTest 和 ArgumentsObjTest。
8. autoRuntimeDialect：默认值为 false。设置为 true 时，允许在运行时根据多数据源自动识别对应方言的分页 （不支持自动选择sqlserver2012，只能使用sqlserver），用法和注意事项参考下面的场景五。
9. closeConn：默认值为 true。当使用运行时动态数据源或没有设置 helperDialect 属性自动获取数据库类型时，会自动获取一个数据库连接， 通过该属性来设置是否关闭获取的这个连接，默认true关闭，设置为 false 后，不会关闭获取的连接，这个参数的设置要根据自己选择的数据源来决定。

重要提示：

当 offsetAsPageNum=false 的时候，由于 PageNum 问题，RowBounds查询的时候 reasonable 会强制为 false。使用 PageHelper.startPage 方法不受影响。

**4. 如何选择配置这些参数**

单独看每个参数的说明可能是一件让人不爽的事情，这里列举一些可能会用到某些参数的情况。

**场景一**

如果你仍然在用类似ibatis式的命名空间调用方式，你也许会用到rowBoundsWithCount， 分页插件对RowBounds支持和 MyBatis 默认的方式是一致，默认情况下不会进行 count 查询，如果你想在分页查询时进行 count 查询， 以及使用更强大的 PageInfo 类，你需要设置该参数为 true。

注： PageRowBounds 想要查询总数也需要配置该属性为 true。

**场景二**

如果你仍然在用类似ibatis式的命名空间调用方式，你觉得 RowBounds 中的两个参数 offset,limit 不如 pageNum,pageSize 容易理解， 你可以使用 offsetAsPageNum 参数，将该参数设置为 true 后，offset会当成 pageNum使用，limit 和 pageSize 含义相同。

**场景三**

如果觉得某个地方使用分页后，你仍然想通过控制参数查询全部的结果，你可以配置 pageSizeZero 为 true， 配置后，当 pageSize=0 或者 RowBounds.limit = 0 就会查询出全部的结果。

**场景四**

如果你分页插件使用于类似分页查看列表式的数据，如新闻列表，软件列表， 你希望用户输入的页数不在合法范围（第一页到最后一页之外）时能够正确的响应到正确的结果页面， 那么你可以配置 reasonable 为 true，这时如果 pageNum<=0 会查询第一页，如果 pageNum>总页数 会查询最后一页。

**场景五**

如果你在 Spring 中配置了动态数据源，并且连接不同类型的数据库，这时你可以配置 autoRuntimeDialect 为 true，这样在使用不同数据源时，会使用匹配的分页进行查询。 这种情况下，你还需要特别注意 closeConn 参数，由于获取数据源类型会获取一个数据库连接，所以需要通过这个参数来控制获取连接后，是否关闭该连接。 默认为 true，有些数据库连接关闭后就没法进行后续的数据库操作。而有些数据库连接不关闭就会很快由于连接数用完而导致数据库无响应。所以在使用该功能时，特别需要注意你使用的数据源是否需要关闭数据库连接。

当不使用动态数据源而只是自动获取 helperDialect 时，数据库连接只会获取一次，所以不需要担心占用的这一个连接是否会导致数据库出错，但是最好也根据数据源的特性选择是否关闭连接。

**3. 如何在代码中使用**

阅读前请注意看[重要提示](https://github.com/pagehelper/Mybatis-PageHelper/blob/master/wikis/zh/Important.md)

分页插件支持以下几种调用方式：

1. *//第一种，RowBounds方式的调用*
2. List<Country> list = sqlSession.selectList("x.y.selectIf", null, new RowBounds(0, 10));
4. *//第二种，Mapper接口方式的调用，推荐这种使用方式。*
5. PageHelper.startPage(1, 10);
6. List<Country> list = countryMapper.selectIf(1);
8. *//第三种，Mapper接口方式的调用，推荐这种使用方式。*
9. PageHelper.offsetPage(1, 10);
10. List<Country> list = countryMapper.selectIf(1);
12. *//第四种，参数方法调用*
13. *//存在以下 Mapper 接口方法，你不需要在 xml 处理后两个参数*
14. public interface CountryMapper {
15. List<Country> selectByPageNumSize(
16. @Param("user") User user,
17. @Param("pageNum") int pageNum,
18. @Param("pageSize") int pageSize);
19. }
20. *//配置supportMethodsArguments=true*
21. *//在代码中直接调用：*
22. List<Country> list = countryMapper.selectByPageNumSize(user, 1, 10);
24. *//第五种，参数对象*
25. *//如果 pageNum 和 pageSize 存在于 User 对象中，只要参数有值，也会被分页*
26. *//有如下 User 对象*
27. public class User {
28. *//其他fields*
29. *//下面两个参数名和 params 配置的名字一致*
30. private Integer pageNum;
31. private Integer pageSize;
32. }
33. *//存在以下 Mapper 接口方法，你不需要在 xml 处理后两个参数*
34. public interface CountryMapper {
35. List<Country> selectByPageNumSize(User user);
36. }
37. *//当 user 中的 pageNum!= null && pageSize!= null 时，会自动分页*
38. List<Country> list = countryMapper.selectByPageNumSize(user);
40. *//第六种，ISelect 接口方式*
41. *//jdk6,7用法，创建接口*
42. Page<Country> page = PageHelper.startPage(1, 10).doSelectPage(new ISelect() {
43. @Override
44. public void doSelect() {
45. countryMapper.selectGroupBy();
46. }
47. });
48. *//jdk8 lambda用法*
49. Page<Country> page = PageHelper.startPage(1, 10).doSelectPage(()-> countryMapper.selectGroupBy());
51. *//也可以直接返回PageInfo，注意doSelectPageInfo方法和doSelectPage*
52. pageInfo = PageHelper.startPage(1, 10).doSelectPageInfo(new ISelect() {
53. @Override
54. public void doSelect() {
55. countryMapper.selectGroupBy();
56. }
57. });
58. *//对应的lambda用法*
59. pageInfo = PageHelper.startPage(1, 10).doSelectPageInfo(() -> countryMapper.selectGroupBy());
61. *//count查询，返回一个查询语句的count数*
62. long total = PageHelper.count(new ISelect() {
63. @Override
64. public void doSelect() {
65. countryMapper.selectLike(country);
66. }
67. });
68. *//lambda*
69. total = PageHelper.count(()->countryMapper.selectLike(country));

下面对最常用的方式进行详细介绍

**1). RowBounds方式的调用**

List<Country> list = sqlSession.selectList("x.y.selectIf", null, new RowBounds(1, 10));

使用这种调用方式时，你可以使用RowBounds参数进行分页，这种方式侵入性最小，我们可以看到，通过RowBounds方式调用只是使用了这个参数，并没有增加其他任何内容。

分页插件检测到使用了RowBounds参数时，就会对该查询进行物理分页。

关于这种方式的调用，有两个特殊的参数是针对 RowBounds 的，你可以参看上面的 场景一 和 场景二

注：不只有命名空间方式可以用RowBounds，使用接口的时候也可以增加RowBounds参数，例如：

1. *//这种情况下也会进行物理分页查询*
2. List<Country> selectAll(RowBounds rowBounds);

注意： 由于默认情况下的 RowBounds 无法获取查询总数，分页插件提供了一个继承自 RowBounds 的 PageRowBounds，这个对象中增加了 total 属性，执行分页查询后，可以从该属性得到查询总数。

2). PageHelper.startPage 静态方法调用

除了 PageHelper.startPage 方法外，还提供了类似用法的 PageHelper.offsetPage 方法。

在你需要进行分页的 MyBatis 查询方法前调用 PageHelper.startPage 静态方法即可，紧跟在这个方法后的第一个MyBatis 查询方法会被进行分页。

**例一：**

1. *//获取第1页，10条内容，默认查询总数count*
2. PageHelper.startPage(1, 10);
3. *//紧跟着的第一个select方法会被分页*
4. List<Country> list = countryMapper.selectIf(1);
5. assertEquals(2, list.get(0).getId());
6. assertEquals(10, list.size());
7. *//分页时，实际返回的结果list类型是Page<E>，如果想取出分页信息，需要强制转换为Page<E>*
8. assertEquals(182, ((Page) list).getTotal());

**例二：**

1. *//request: url?pageNum=1&pageSize=10*
2. *//支持 ServletRequest,Map,POJO 对象，需要配合 params 参数*
3. PageHelper.startPage(request);
4. *//紧跟着的第一个select方法会被分页*
5. List<Country> list = countryMapper.selectIf(1);
7. *//后面的不会被分页，除非再次调用PageHelper.startPage*
8. List<Country> list2 = countryMapper.selectIf(null);
9. *//list1*
10. assertEquals(2, list.get(0).getId());
11. assertEquals(10, list.size());
12. *//分页时，实际返回的结果list类型是Page<E>，如果想取出分页信息，需要强制转换为Page<E>，*
13. *//或者使用PageInfo类（下面的例子有介绍）*
14. assertEquals(182, ((Page) list).getTotal());
15. *//list2*
16. assertEquals(1, list2.get(0).getId());
17. assertEquals(182, list2.size());

例三，使用PageInfo的用法：

1. *//获取第1页，10条内容，默认查询总数count*
2. PageHelper.startPage(1, 10);
3. List<Country> list = countryMapper.selectAll();
4. *//用PageInfo对结果进行包装*
5. PageInfo page = new PageInfo(list);
6. *//测试PageInfo全部属性*
7. *//PageInfo包含了非常全面的分页属性*
8. assertEquals(1, page.getPageNum());
9. assertEquals(10, page.getPageSize());
10. assertEquals(1, page.getStartRow());
11. assertEquals(10, page.getEndRow());
12. assertEquals(183, page.getTotal());
13. assertEquals(19, page.getPages());
14. assertEquals(1, page.getFirstPage());
15. assertEquals(8, page.getLastPage());
16. assertEquals(true, page.isFirstPage());
17. assertEquals(false, page.isLastPage());
18. assertEquals(false, page.isHasPreviousPage());
19. assertEquals(true, page.isHasNextPage());

**3). 使用参数方式**

想要使用参数方式，需要配置 supportMethodsArguments 参数为 true，同时要配置 params 参数。 例如下面的配置：

1. <plugins>
2. *<!-- com.github.pagehelper为PageHelper类所在包名 -->*
3. <plugin interceptor="com.github.pagehelper.PageInterceptor">
4. *<!-- 使用下面的方式配置参数，后面会有所有的参数介绍 -->*
5. <property name="supportMethodsArguments" value="true"/>
6. <property name="params" value="pageNum=pageNumKey;pageSize=pageSizeKey;"/>
7. </plugin>
8. </plugins>

在 MyBatis 方法中：

1. List<Country> selectByPageNumSize(
2. @Param("user") User user,
3. @Param("pageNumKey") int pageNum,
4. @Param("pageSizeKey") int pageSize);

当调用这个方法时，由于同时发现了 pageNumKey 和 pageSizeKey 参数，这个方法就会被分页。params 提供的几个参数都可以这样使用。

除了上面这种方式外，如果 User 对象中包含这两个参数值，也可以有下面的方法：

List<Country> selectByPageNumSize(User user);

当从 User 中同时发现了 pageNumKey 和 pageSizeKey 参数，这个方法就会被分页。

注意：pageNum 和 pageSize 两个属性同时存在才会触发分页操作，在这个前提下，其他的分页参数才会生效。

**3). PageHelper 安全调用**

**1. 使用 RowBounds 和 PageRowBounds 参数方式是极其安全的**

**RowBounds：行范围**

**2. 使用参数方式是极其安全的**

**3. 使用 ISelect 接口调用是极其安全的**

ISelect 接口方式除了可以保证安全外，还特别实现了将查询转换为单纯的 count 查询方式，这个方法可以将任意的查询方法，变成一个 select count(\*) 的查询方法。

**4. 什么时候会导致不安全的分页？**

PageHelper 方法使用了静态的 ThreadLocal 参数，分页参数和线程是绑定的。

只要你可以保证在 PageHelper 方法调用后紧跟 MyBatis 查询方法，这就是安全的。因为 PageHelper 在 finally 代码段中自动清除了 ThreadLocal 存储的对象。

如果代码在进入 Executor 前发生异常，就会导致线程不可用，这属于人为的 Bug（例如接口方法和 XML 中的不匹配，导致找不到 MappedStatement 时）， 这种情况由于线程不可用，也不会导致 ThreadLocal 参数被错误的使用。

但是如果你写出下面这样的代码，就是不安全的用法：

1. PageHelper.startPage(1, 10);
2. List<Country> list;
3. if(param1 != null){
4. list = countryMapper.selectIf(param1);
5. } else {
6. list = new ArrayList<Country>();
7. }

这种情况下由于 param1 存在 null 的情况，就会导致 PageHelper 生产了一个分页参数，但是没有被消费，这个参数就会一直保留在这个线程上。当这个线程再次被使用时，就可能导致不该分页的方法去消费这个分页参数，这就产生了莫名其妙的分页。

上面这个代码，应该写成下面这个样子：

1. List<Country> list;
2. if(param1 != null){
3. PageHelper.startPage(1, 10);
4. list = countryMapper.selectIf(param1);
5. } else {
6. list = new ArrayList<Country>();
7. }

这种写法就能保证安全。

如果你对此不放心，你可以手动清理 ThreadLocal 存储的分页参数，可以像下面这样使用：

1. List<Country> list;
2. if(param1 != null){
3. PageHelper.startPage(1, 10);
4. try{
5. list = countryMapper.selectAll();
6. } finally {
7. PageHelper.clearPage();
8. }
9. } else {
10. list = new ArrayList<Country>();
11. }

这么写很不好看，而且没有必要。