

38 | 分页的那些事儿

2019-12-06 四火

全栈工程师修炼指南

[进入课程 >](#)



讲述：四火

时长 16:21 大小 11.24M



你好，我是四火。

分页是全栈开发中非常常见的一个功能，绝大多数网站都需要用到。这个功能可谓麻雀虽小五脏俱全，是从呈现层、控制器层、到模型层，并和数据存储密切相关的真正的“全栈”功能。有时候你能见到一些清晰、明确的分页，也就是说，你能看到某一个内容的呈现被分成若干页，而当前又是在第几页；而有时候这个功能则是部分模糊了的，比方说，能翻页，但是并不显示总共有多少页。那今天，就让我们了解一些典型的分页方法，理解分页的一些常见问题和它们的解决方式。

物理分页和逻辑分页

物理分页一般指的就是数据库分页，而逻辑分页则是程序员口中的“内存分页”。前者是通过条件查询的方式，从数据库中取出特定的数据段来，而后者，则是在完整数据都加载在内存中的情况下，从中取出特定段的数据来。

显然我们一般情况下更关注物理分页，因为内存的大小总是更为有限，多数情况下我们希望通过条件查询的方式去数据库中取出特定页来。但是也有反例，比方说某些数据量不大，但是访问频率较高的数据，以“缓存”的形式预加载到内存中，再根据客户端的分页条件返回数据，这种方式一样可以遵循分页接口，但是由于被分页的数据是全部在内存中的，这样的数据一般需要遵循如下几个要求：

数据量不大，可以放在内存中；

一致性满足要求，或者使用专门的设计维护内存中数据的一致性；

持久性方面，数据允许丢失，因为在内存中，断电即丢失。

分页代码设计

分页这个事儿从功能抽象的角度看基本不复杂，但是我们也经常能看到不同的实现。比较常见的有两种，指定页码的分页，以及使用 token 的分页，当然，严格来说，页码其实是 token 的一种特殊形式。就以指定页码的分页为例，在实际代码层面上，我见到的分页代码设计包括这样两种。

单独的分页工具类

第一种是彻底把 DAO 的查询逻辑拿出去，创建一个单独的分页工具类 Paginator。这个工具的实例对象只负责分页状态的存储和变更，不知道也不管理数据，在实例化的时候需要 totalCount 和 pageSize 两个参数，前者表示总共有多少条数据，后者表示每一页最多显示多少条。

DAO 的查询接口自己去创建或者接纳一个 Paginator 对象，其实现会调用对象的 getStart() 和 getEnd() 方法，从而知道查询时数据的范围。请看这个 Paginator 的例子，我把方法内的具体实现省略了：

 复制代码

```
1 class Paginator {  
2     public Paginator(int totalCount, int pageSize) {}  
3 }
```

```


4      // 翻页
5      public void turnToNextPage() {}
6      public void turnToPrevPage() {}
7      public void setPageNo(int pageNo) {}
8
9      // 数据访问层需要的 start 和 end
10     public int getStart() {}
11     public int getEnd() {}
12
13     // 是否还有上一页、下一页
14     public boolean hasNextPage() {}
15     public boolean hasPrevPage() {}
16
17     // 当前在哪一页
18     public int getPageNo() {}
19 }

```

绑定查询结果的分页对象

第二种则将分页对象和查询数据的结果集绑定在一起，有的是将结果集对象继承自一个分页类，有的则是将结果集对象以组合的方式放到分页对象里面，数据访问层直接返回这个携带实际数据的数据的分页对象。从优先使用组合而不是继承的角度来说，组合的方式通常更好一些。

具体说，我们可以定义这样的分页对象：

 复制代码

```

1  public class Page<T> {
2      private int pageNo;
3      private int totalCount;
4      private int pageSize;
5      private Collection<T> data; // 当前页的实际业务对象
6
7      public Page(Collection<T> data, int pageSize, int pageNo, int totalCount) {
8          this.data = data;
9          this.pageSize = pageSize;
10         this.pageNo = pageNo;
11         this.totalCount = totalCount;
12     }
13
14     public int getPageSize() {}
15     public int getPageNo() {}
16     public int getTotalPages() {}
17     public int getLastPageNo() {}
18     public Collection<T> getData() {}
19 }

```

你看，这个分页对象和前面的那个例子不同，它将 DAO（数据访问对象，在 [🔗\[第 12 讲\]](#) 有介绍）的某次查询的结果，也就是某一页上的数据集合，给包装起来了。因此这个对象是由 DAO 返回的，并且里面的数据仅仅是特定某一页的，这也是它没有提供用于翻页的方法的原因——具体要访问哪一页的数据，需要将页码和页面大小这样的信息传给 DAO 去处理。

其它方法

比如说，如果按照充血模型的方式来设计（关于充血模型是什么，以及代码实现的例子，你可以参见 [🔗\[第 08 讲\]](#)），则可以**把这个 Page 对象和 DAO 的代码整合起来，放到同一个有状态的业务对象上面去，这样这个对象既携带了业务数据，又携带了分页信息**。当然，这个方法的区分其实只是贫血模型和充血模型的区别，从分页设计的角度上来看，和上面的第二种其实没有本质区别。

此外，一些“管得宽”的持久层 ORM 框架，往往也把分页的事情给包装了，给开发人员开放易于使用的分页接口。比方说 Hibernate，你看这样一个例子：

 复制代码

```
1 session
2 .createCriteria(Example.class)
3 .setFirstResult(10)
4 .setMaxResults(5)
5 .list();
```

这就是通过 Hibernate 的 Criteria Query API 来实现的，代码也很易于理解。当然，如果想要取得总行数，那可以用 Projection：

 复制代码

```
1 session
2 .createCriteria(Example.class)
3 .setProjection(Projections.rowCount())
4 .list()
5 .get(0);
```

SQL 实现

如果使用关系数据库，那么落实到 SQL 上，以 MySQL 为例，大致是这样的：

```
1 select * from TABLE_NAME limit 3, 4;
```

limit 后面有两个参数，第一个表示从下标为 3 的行开始返回，第二个表示一共最多返回 4 行数据。

但是，如果使用的是 Oracle 数据库，很多朋友都知道可以使用行号 rownum，可是这里面有一个小坑——Oracle 给 rownum 赋值发生在当条子句的主干执行完毕后，比如说：

```
1 select * from TABLE_NAME where rownum between 2 and 4;
```

如果表里面有 100 行数据，那么你觉得上面那条 SQL 会返回什么？

返回 2 行到第 4 行的数据吗？不对，它什么都不会返回。

这是为什么呢？因为对每一行数据，执行完毕以后才判断它的 rownum，于是：

第一行执行后的 rownum 就是 1，而因为不符合“2 到 4”的条件，因此该行遍历后，就从结果集中抛弃了；

于是执行第二行，由于前面那行被抛弃了，那这一行是新的—行，因此它的 rownum 还是 1，显然还是不符合条件，抛弃；

每一行都这样操作，结果就是，所有行都被抛弃，因此什么数据都没有返回。

解决方法也不难，用一个嵌套就好，让里面那层返回从第 1 条到第 4 条数据，并给 rownum 赋一个新的变量名 r，外面那层再过滤，取得 r 大于等于 3 的数据：

```
1 select * from (select e.*, rownum r from TABLE_NAME e where rownum<=4) where r:
```


重复数据的问题

下面我们再看一个分页的常见问题——不同页之间的重复数据。

这个问题指的是，某一条数据，在查询某一个接口第一页的时候，就返回了；可是查询第二页的时候，又返回了。

我来举个例子：

```
1 select NAME, DESC from PRODUCTS order by COUNT desc limit 0, 50;
```

 复制代码

这个查询是要根据产品的数量倒序排序，然后列出第一页（前 50 条）来。

接着查询第二页：

```
1 select NAME, DESC from PRODUCTS order by COUNT desc limit 50, 50;
```

 复制代码

结果发现有某一个产品数据，同时出现在第一页和第二页上。按理说，这是不应当发生的，这条数据要么在第一页，要么在第二页，怎么能同时在第一页和第二页上呢？


其实，这只是问题现象，而其中的原因是不确定的，比如说有这样两种可能。

排序不稳定

第一种是因为查询的结果排序不稳定。

说到这个“不稳定”，其实本质上就是排序算法的“不稳定”。如果对算法中的排序算法比较熟悉的话，你应该知道，**我们说排序算法稳不稳定，说的是当两个元素对排序来说“相等”的时候，是不是能够维持次序和排序前一样。**比如有这样一组数：

```
1 1, 2a, 5, 2b, 3
```

 复制代码

这里面有两个 2，但是为了区分，其中一个叫做 2a，另一个叫做 2b。现在 2a 在 2b 前面，如果排序以后，2a 还是能够保证排在 2b 前面，那么这个排序就是稳定的，反之则是不稳定的。有很多排序是稳定的，比如冒泡排序、插入排序，还有归并排序；但也有很多是不稳定的，比如希尔排序、快排和堆排序。

再回到我们的问题上，分页查询是根据 COUNT 来排序的，如果多条数据，它们的 COUNT 一样，而在某些数据库（包括某些特定版本）下，查询的排序是不稳定的。这样就可能出现，这个相同 COUNT 的产品记录，在结果集中出现的顺序不一致的问题，那也就可能出现某条记录在第一页的查询中出现过了，而在第二页的查询中又出现了一遍。

其实，从数据库的角度来说，在我们根据上述代码要求数据库返回数据的时候，COUNT 相同的情况下，可并没有要求数据库一定要严格遵循某一个顺序啊，因此严格说起来，数据库这么做本身也是没有什么不对的。

无论如何，问题如果明确是这一个，那么解决方法就比较清楚了。既然问题只会在 COUNT 相同的时候出现，那么上例中，我们给 order by 增加一个次要条件——ID，而因为 ID 是表的主键，不可能出现重复，因此在 COUNT 相同的时候排序一定是严格按照 ID 来递减的，这样也就可以保证排序不会因为“不稳定”而造成问题：

 复制代码

```
1 select NAME, DESC from PRODUCTS order by COUNT, ID desc limit 0, 50;
```

数据本身变化

第二个造成重复数据问题的原因是数据本身的变化。

这个也不难理解，比如还是上面这行 SQL：

 复制代码

```
1 select NAME, DESC from PRODUCTS order by COUNT, ID desc limit 0, 50;
```

本来有一行数据是在第二页的开头，用户在上述查询执行后，这行数据突然发生了变化，COUNT 增加了不少，于是挤到第一页去了，那么相应地，第一页的最后一条数据就被挤到第二页了，于是这时候用户再来查询第二页的数据：

```
1 select NAME, DESC from PRODUCTS order by COUNT, ID desc limit 50, 50;
```

这就发现原来第一页尾部的一条数据，又出现在了第二页的开头。

对于这个问题，我们也来看看有哪些解决方案。但是在看解决方案之前，我们先要明确，这是不是一个非得解决的问题。换言之，如果产品经理和程序员都觉得重复数据并不是什么大不了的事情，这个问题就可以放一放，我们不需要去解决那些“可以不是问题”（或者说优先极低）的问题。我知道很多人觉得这个太过浅显，甚至不值得讨论，但毕竟这是一个很基本的原则，做技术的我们在面对问题的时候始终需要明确，而不是一头扎进解决问题的泥塘里出不来了。

至于解决方案，比较常见的包括这样几个：

1. 结果过滤

如果我们认定两次查询的结果中，可能出现重复，但是考虑到数据变更的速度是比较慢的，顺序的变化也是缓慢的，因此这个重复数据即便有，也会很少。那么，第二页展示的时候，我们把结果中第一页曾经出现过的这些个别的数据给干掉，这样就规避了数据重复的问题。

这大概是最简单的一种处理方式，但是其不足也是很明显的：

说是“个别数据”，可到底有多少重复，这终究存在不可预测性，极端情况下第二页的数据可能会出现大量和第一页重复的情况，删除这些重复数据会导致第二页数据量特别少，从而引发糟糕的用户体验；

数据丢失问题：既然第二页上出现第一页的重复，那就意味着存在某数据在用户查询第一页的时候它待在第二页，而用户查询第二页的时候又跑到第一页上去了，这样的数据最终没有返回给用户；

有第一页、第二页，那就还有第三页、第四页的问题，比如第一页的数据可能跟第四页重复啊，如果我们把这些因素都考虑进去，这个方案就没有那么“简单”了。

2. 独立的排序版本

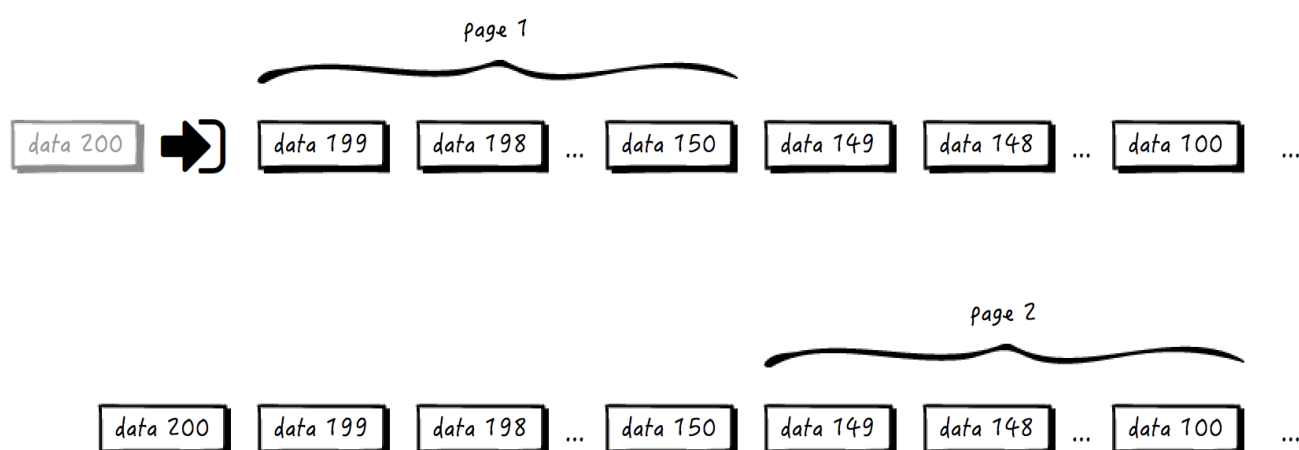
这个方法原理上也很简单，对于任何排序，都维持一个版本号。这样，在数据产生变化的时候，这个新的排序，需要采用一个不同的版本号。本质上，这就是把每次排序都单独拿出来维护，每一个排序都有一份独立的拷贝。

这种方法适合这个排序变更不太频繁的情况，因为它的缺点很明显，就是要给每一份排序单独存放一份完整的拷贝。但是，它的优点也很明显，就是在多次查询的过程中，这个列表是静态的，不会改变的，不需要担心数据重复和丢失的问题。特别是对于开放的 Web API 来说，我们经常需要用一个循环多次查询才能获取全量的数据，这个机制就很有用了。

3. 数据队列

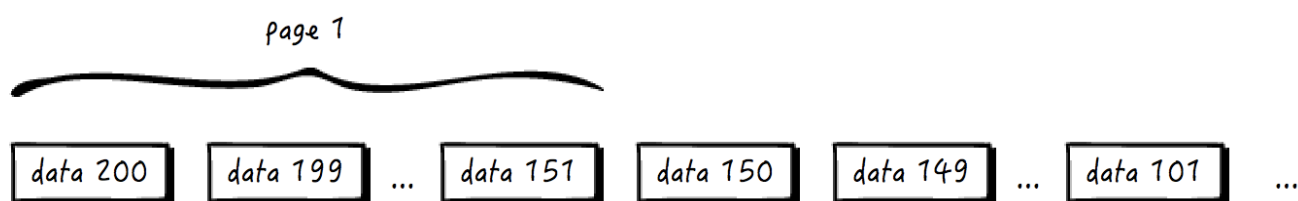
在某些场景下，我们如果能保证数据的顺序不变，而添加的数据总在显示列表的一头，也就是说，把原本的数据集合变成一个队列，这样要解决重复数据问题的时候，就会有比较好的解决办法了。

每次查询的时候，都记住了当前页最后一条记录的位置（比如我们可以使用自增长的 ID，或是使用数据添加时的 timestamp 来作为它“位置”的标记），而下一页，就从该记录开始继续往后查找就好了。这样无论是否有新添加的数据，后面页数的切换，使用的都是相对位置，也就不会出现数据重复的问题了。看下面的例子：



你看，用户刚访问的时候，返回了 data 150 到 data 199 这 50 条记录，并且记住了当前页尾的位置。用户再访问第二页的时候，其实已经有新的元素 data 200 加进来了，但是不管它，我们根据前一页的页尾的位置找到了第二页开头的位置，返回 data 100 到 data 149 这 50 条记录。

当然，只有一种例外，即只有用户访问第一页的时候（或者说，是用户的查询传入的“位置”参数为空的时候），才始终查询并返回最新的第一页数据。比如现在 data 200 已经添加进来了，查询第一页的时候就返回 data 151 到 data 200 这最新的 50 条数据：



上述这种方式其实也挺常见的，特别像是新闻类网站（还有一些 SNS 网站），基本上一个栏目下新闻的发布都可以遵循这个队列的规则，即新发布的新闻总是可以放在最开始的位置，发布以后的新闻相对位置不发生改变。还有就是许多 NoSQL 数据库用于查询特定页的 token，都是利用了这个机制。

总结思考

今天我们一起学习了分页的前前后后，既包括设计、实现，也包括对分页常见的重复数据的问题的分析，希望今天的内容能让你有所收获。

老规矩，我来提两个问题吧：

对于文中介绍的工具类 `Paginator`，我把方法签名写出来了，你能把这个类实现完整吗？

对于“分页代码设计”一节中介绍的两种分页对象的设计方法，它们各有优劣，你能比较一下吗？

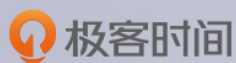
好，今天就到这里，欢迎你在留言区和我讨论。

扩展阅读

对于 Hibernate 分页查询用到的条件查询 API，你可以阅读 [这篇文章](#)，看到更多的例子；而对于通过 Projection 来取得总行数的代码，[这里有完整例子](#)。

本讲一开始的时候我提到，分页的设计上，有的是使用指定页码的，也是本讲的重点；还有一种是使用 token 的，这种方式也是很多 Web API 常提供的方式，你可以阅读

DynamoDB 的 [📄 官方文档](#) 了解一下它的分页 API。



全栈工程师修炼指南

从全栈入门到技能实战

熊燚

Oracle 首席软件工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 37 | 全栈开发中的算法（下）

下一篇 39 | XML、JSON、YAML比较

精选留言 (4)

 写留言




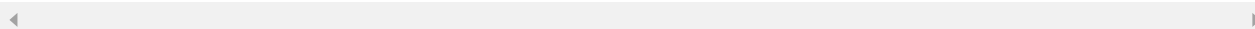
springday

2019-12-07

解决了困扰了我多年的问题。

展开 ▾

作者回复: 



靠人品去赢

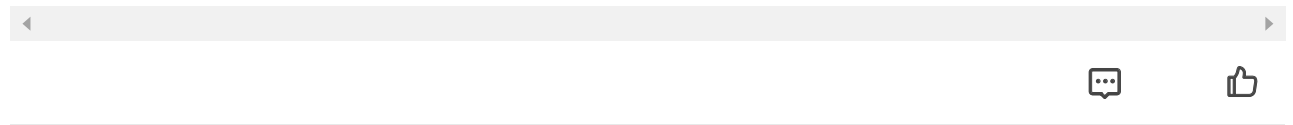
2019-12-06

刚想说重复数据的问题，就提了。

我们遇到情况就是，订单列表无所谓不是第一页，每次到页面触发按钮的时候直接请求最新的数据，量不多可以忽略。

但是首页的展示或者用的比较频繁就不能，这样看起来很怪，之前虎嗅的网页版具有类似的问题，下拉新的一页和上面的数据有相同的。我们的想法是，拿到前一页的最后一个id...
展开 ▾

作者回复: "最后一个 id"，就是文中我说的“数据队列”的方式。



许童童

2019-12-06

老师这一节讲的第三种方法，数据队列正是我在业务中使用到的，很精妙，很好的解决了问题



leslie

2019-12-06

数据库端做limit操作前端页面调用结果集这大概是最方面且最快捷的方式：调用的算法和数据结构同样简单；代价就是资源的消耗。无论是mysql还是sql server分页直接通过类似方式用的最多-时间代价相对小，擅长的东西做事擅长的事情。另外一种方式没尝试过。

展开 ▾

