getArticleList

无特殊参数查询

getArticleList

getTotal优化(TODO)

getTagListByArticleId优化(TODO)

Limit优化(doing)

**关于本博客在本地运行环境下的性能优化。**

博客上线后，可能会被很多人同时访问，因此，在上线前，最好对博客进行一定程度的优化。本博客分为前端项目，后端项目，在本文章中，作者主要探讨、实施了对后端项目的优化。这些优化包括对getArticleList接口，

1. getArticleList接口

getArticleList由4钟分页查询组成：用标签tagId查询，用类别categoryId查询，用内容queryContent查询，以及普通分页查询。

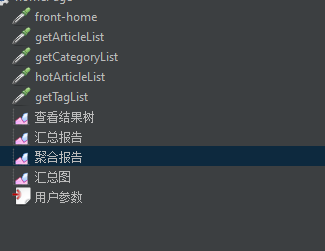
1-1.getArticleList作为普通分页查询接口的性能优化。

* 1. 背景：

用户在进入首页后，会调用很多接口，包括前端工程接口front-home，后端工程接口：普通分页查询getArticleList,getCategoryList,hotArticleList,getTagList。通过jmeter进行测试后，发现getArticleList在进入首页这个功能中占用的时间相对于其他后端工程接口所占用的时间比较长，因此我们考虑对其进行优化。而用户在首页点击“加载更多”按钮后，调用的后端接口只有普通分页查询getArticleList。因此我们在此进行统一优化。

***进入首页功能性能测试如下：***

用50线程数/秒对进入首页进行模拟并发请求，结果如下图：



（内容为模拟进入首页的线程组，其中front-home是对前端服务的取样器，其余为后端取样器）



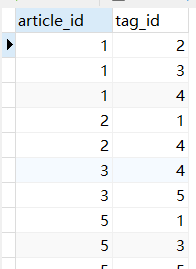
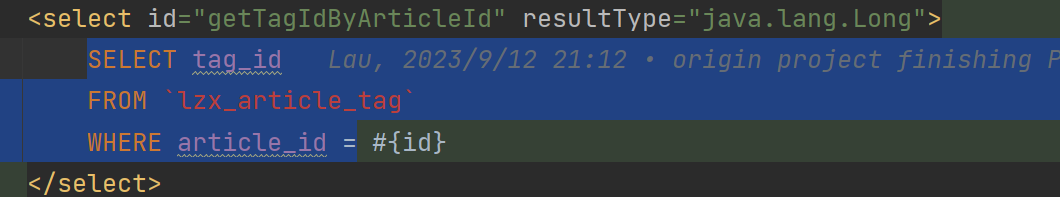
（内容为吞吐量平稳之后，各取样器的相应请求数据）

从上面两图不难看出，在后端接口中，getArticleList接口的平均访问时间为11390ms，中位数为9299ms，在后端接口中明显性能相对不好，因此，优化getArticleList接口成了优化进入首页的重要一环。

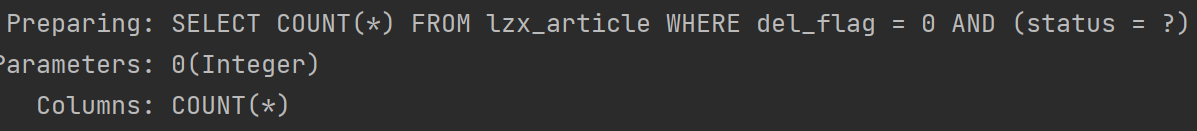
* 1. “进入首页”的getArticleList优化-可以考虑缓存

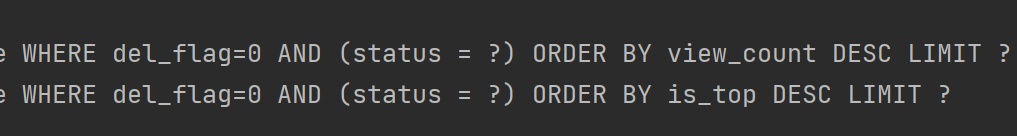
如何优化？“进入首页”的getArticleList接口的代码逻辑都比较简单，主要是从数据库获取内容，因此重点放在优化查询数据库上。同时，不难想到mybatisplus的数据库操作日志也占用性能，因此也要考虑关闭后端mybatisplus日志，当然关闭日志（TODO）可以暂时放一放。

要想优化查询数据库，首先我们需要看进入首页的getArticleList包括哪些查询。从代码不难得知，该接口的查询主要包括getCategoryName(categoryId)，getTagList(articleId),以及分页查询。getCategoryName查询的是根据lzx\_category表的主键进行查询，而这张表又很小（少于100行），所以暂不考虑优化。对于getTagList(articleId),该查询又分为从article\_tag这张2个字段的relation表根据主键查询TagId（如下面的两张图），以及从lzx\_tag这张不大（少于100行）表根据主键查询，所以也暂时不考虑优化。



那么我们就来看分页查询。从前端项目进入首页，然后查看后端日志，如下两图所示。从下两图不难看出，“进入首页”涉及到2条sql查询，其中第一条（TODO，第一条也尝试优化）是用来查询所有记录数。第二条包含del\_flag,status,view\_count,is\_top字段。



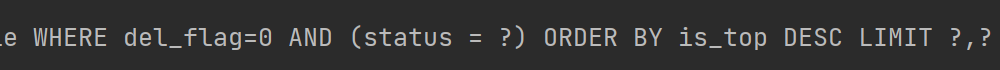


（“进入首页”接口的后端sql语句）

首先看del\_flag,删除字段只有两个值0和1，因此没有必要对此建立索引，status，is\_top同理。因此，想要使用索引优化，并不合适。但是鉴于访问首页的这些article经常访问，而且对于数据一致性要求并不高，我们可以考虑将这些article先放入redis缓存（TODO）中。

* 1. 点击“加载更多”的性能优化

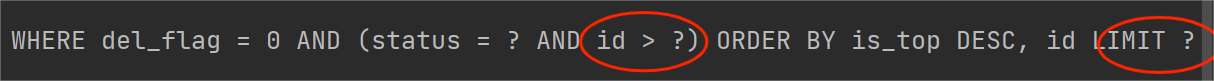
点击“加载更多”按钮的getArticleList代码与“进入首页”一致，因此根据“进入首页”优化的讨论，我们把优化“加载更多”的重点放在优化分页查询。这时的sql仍然为2条，与“进入首页”不同的是，第二条sql语句变为了如下图：



（Sql从limit m变成了 limit m, n）

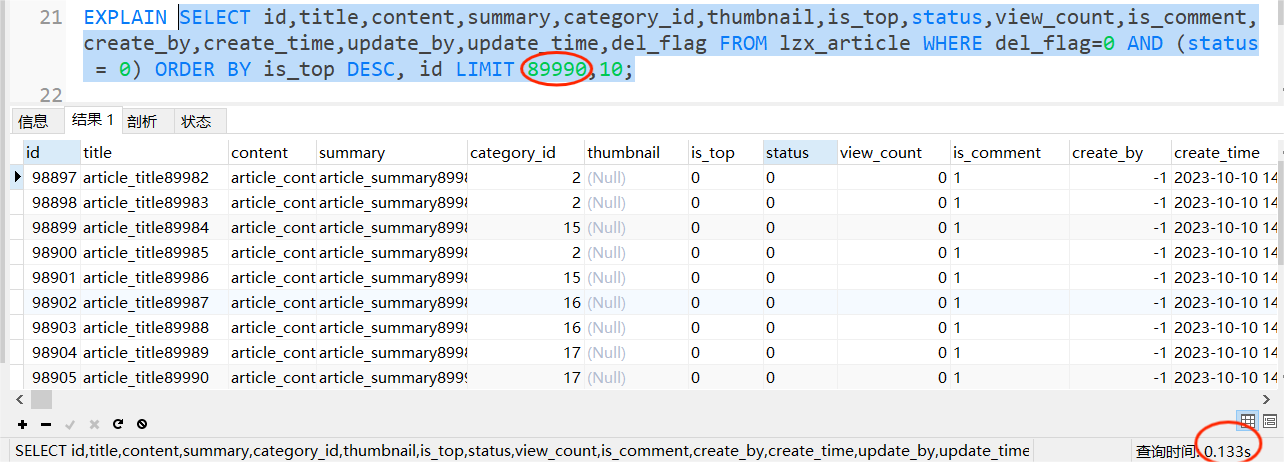
Sql从limit m变成了 limit m, n。这种sql在article数据量特别大时会导致全局扫描太多数据并丢弃，很浪费性能。因此我们对此尝试优化。我们需要自定义分页查询，但这时我发现原代码并不好仅仅针对首页中的getArticleList进行自定义分页查询，因此我对其进行了一定程度的更改。

那么这种情况下如何进行优化？我的想法是，既然利用del\_flag,is\_top,status建立索引都不合适，那么，不妨引入上一页的最大id，利用主键索引先进行范围查找，然后再排序与截取，这样就避免了扫描浪费。于是，更改代码，使原语句变成如下图所示（TODO，无论优不优化，会出现数据重复的问题）。

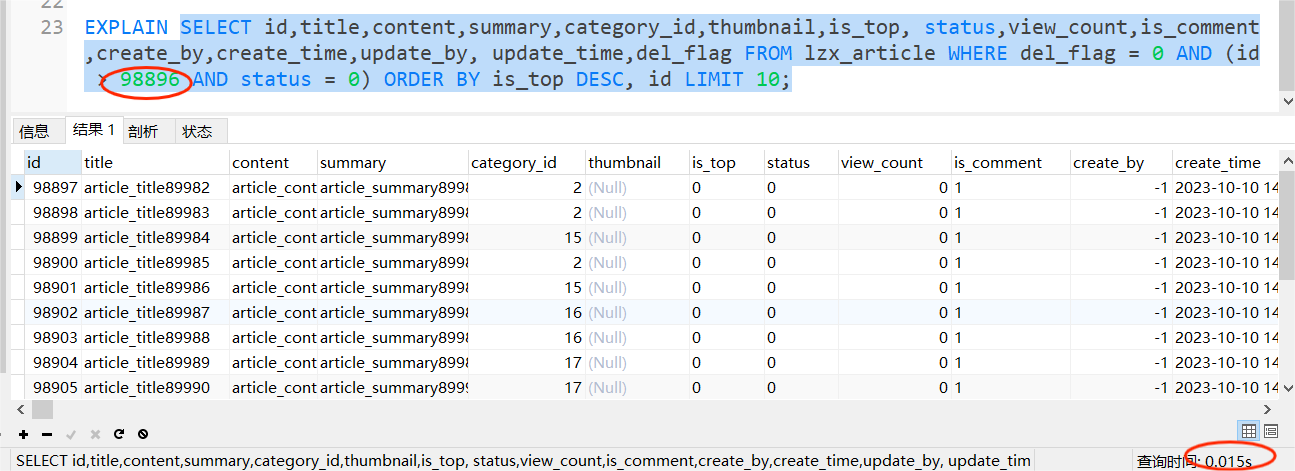


（先用id排序，再进行后续操作）

那么，优化的效果如何？拿查询行数为89991-90000的文章来进行测试，首先来看sql执行时间，从下图来看，不难看出，优化后的sql比优化前的sql在页数比较多时，消耗时间减少了80%多。



（优化前，可以看到查询时间为0.133s）



（优化后，可以看到查询时间缩短为0.015s）

可以看到，sql优化效果不错。那么实际业务优化情况如何？我们在jmeter中，对首页中的“加载更多”进行测试，结果如下图所示，可以看到，当页数比较大时，吞吐量从10/s变成了300/sec。（注：1.经过测试，页数比较小时，两者性能相近，在这里不予展示。2.“加载更多”的业务接口主要是后端的getArticleList，因此本次测试忽略前端接口）



（“首页查询”的“加载更多”测试，上图为接口参数）



（“首页查询”的“加载更多”测试，上图为优化后测试数据）



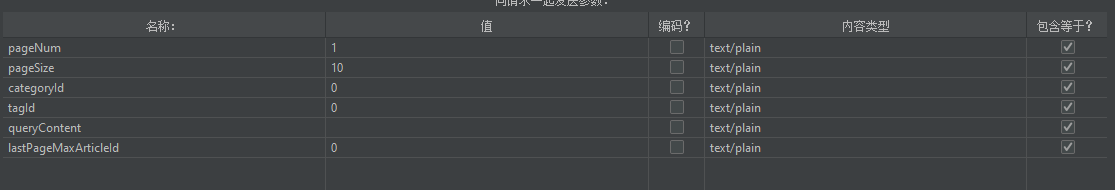
（“首页查询”的“加载更多”测试，上图为优化前测试数据，其中异常量太大是因为请求处理比较慢，很多请求还没收到信息就被停止了）

1-2.getArticleList作为根据tagId查询接口的优化

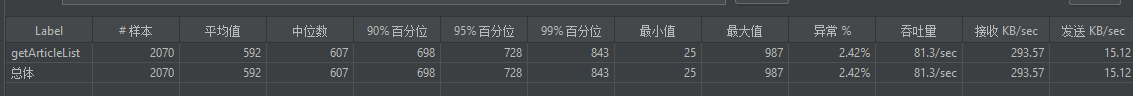
(1) 背景

在文章的tagList或者任一页面右侧的tagList点击某一个tag，会查询有此tag的文章。但此接口涉及到查询lzx\_article\_tag，lzx\_article两张行数比较大的表（都在90000行以上），因此我们考虑对此进行优化。

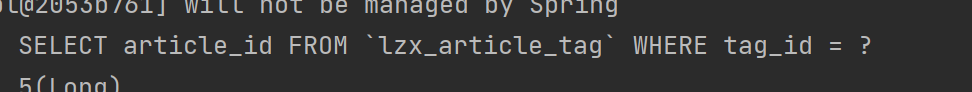
-----------------------------

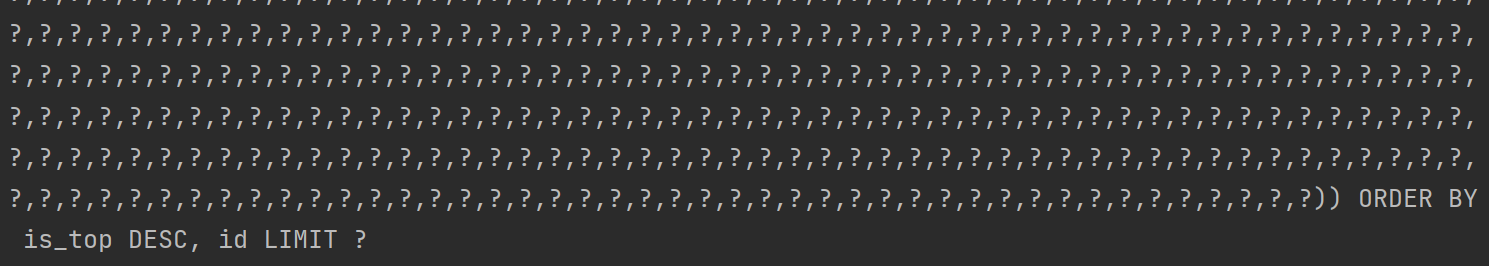




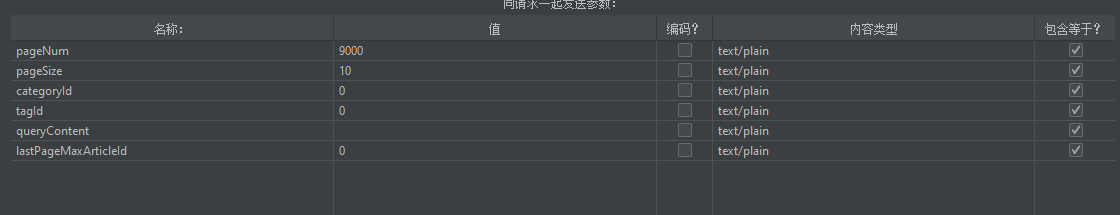


=============





-----------------------------





======================

-------------------

数据量小



=============