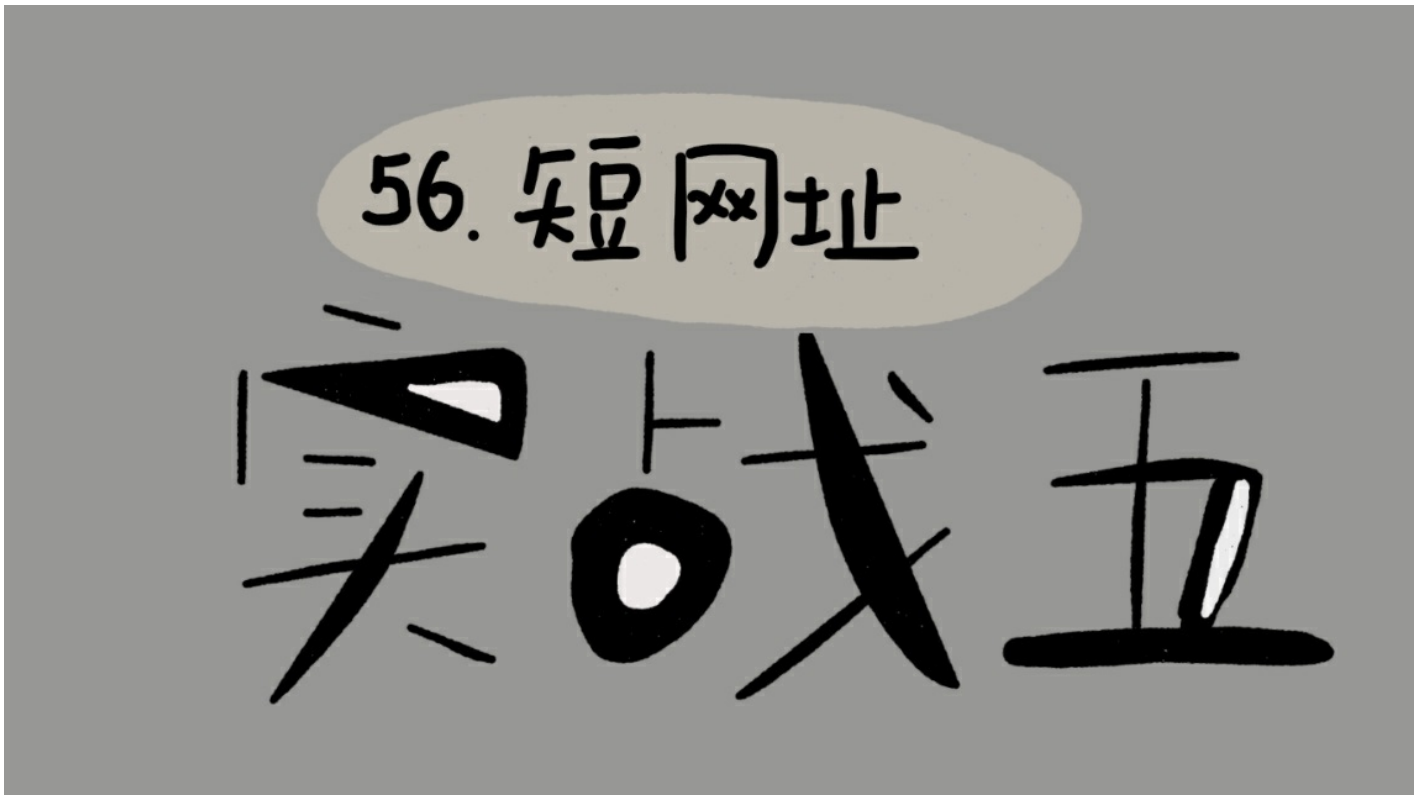
56讲算法实战（五）：如何用学过的数据结构和算法实现一个短网址系统



短网址服务你用过吗？如果我们在微博里发布一条带网址的信息，微博会把里面的网址转化成一个更短的网址。我们只要访问这个短网址，就相当于访问原始的网址。比如下面这两个网址，尽管长度不同，但是都可以跳转到我的一个GitHub开源项目里。其中，第二个网址就是通过新浪提供的短网址服务生成的。

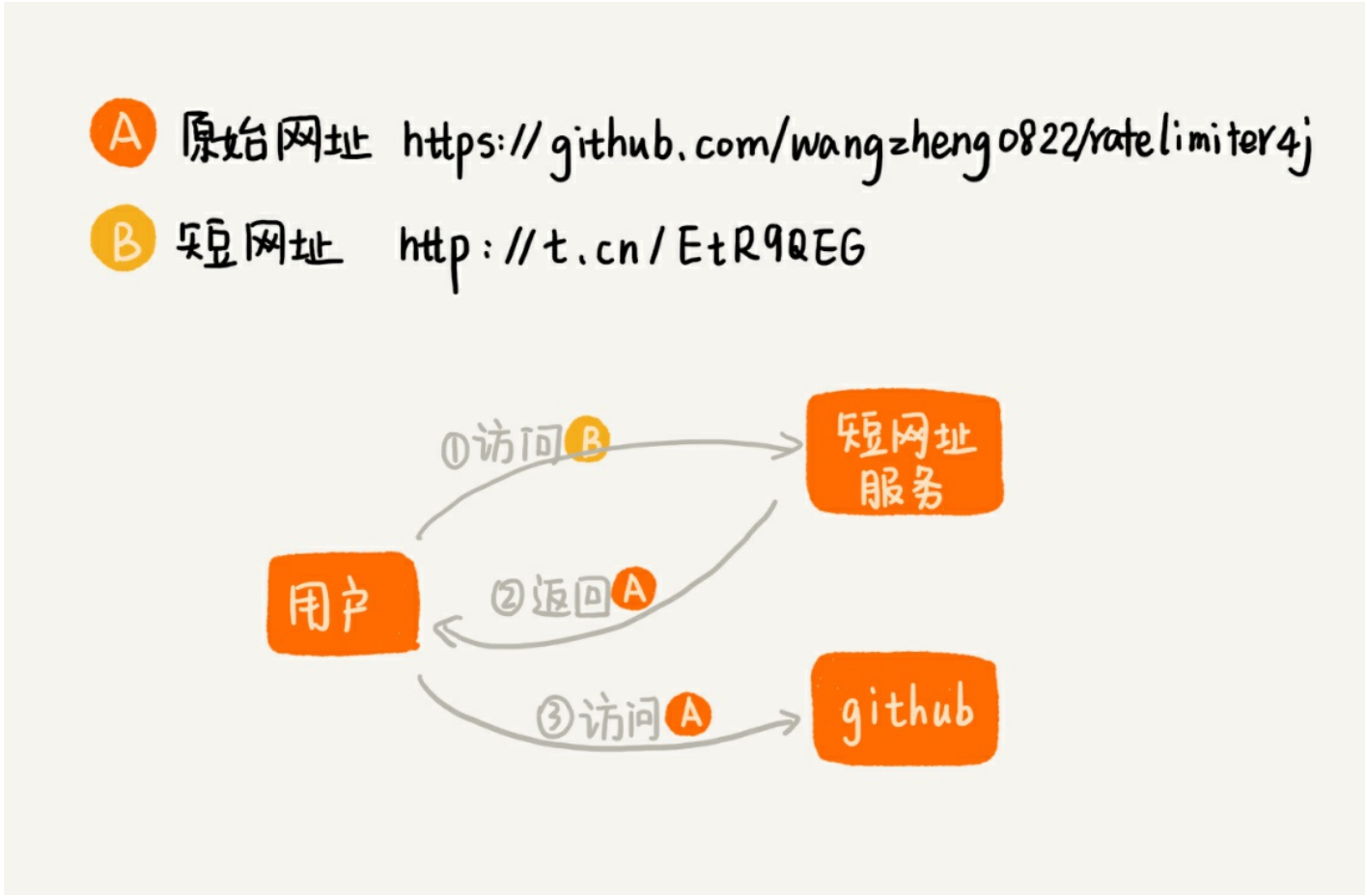
```
原始网址：https://github.com/wangzheng0822/ratelimiter4j
短网址：http://t.cn/EtR9QEG
```

从功能上讲，短网址服务其实非常简单，就是把一个长的网址转化成一个短的网址。作为一名软件工程师，你是否思考过，这样一个简单的功能，是如何实现的呢？底层都依赖了哪些数据结构和算法呢？

短网址服务整体介绍

刚刚我们讲了，短网址服务的一个核心功能，就是把原始的长网址转化成短网址。除了这个功能之外，短网址服务还有另外一个必不可少的功能。那就是，当用户点击短网址的时候，短网址服务会将浏览器重定向为原始网址。这个过程是如何实现的呢？

为了方便你理解，我画了一张对比图，你可以看下。



从图中我们可以看出，浏览器会先访问短网址服务，通过短网址获取到原始网址，再通过原始网址访问到页面。不过这部分功能并不是我们今天要讲的重点。我们重点来看，如何将长网址转化成短网址？

如何通过哈希算法生成短网址？

我们前面学过哈希算法。哈希算法可以将一个不管多长的字符串，转化成一个长度固定的哈希值。我们可以利用哈希算法，来生成短网址。

前面我们已经提过一些哈希算法了，比如MD5、SHA等。但是，实际上，我们并不需要这些复杂的哈希算法。在生成短网址这个问题上，毕竟，我们不需要考虑反向解密的难度，所以我们只需要关心哈希算法的计算速度和冲突概率。

能够满足这样要求的哈希算法有很多，其中比较著名并且应用广泛的一个哈希算法，那就是MurmurHash算法。尽管这个哈希算法在2008年才被发明出来，但现在它已经广泛应用到Redis、MemCache、Cassandra、HBase、Lucene等众多著名的软件中。

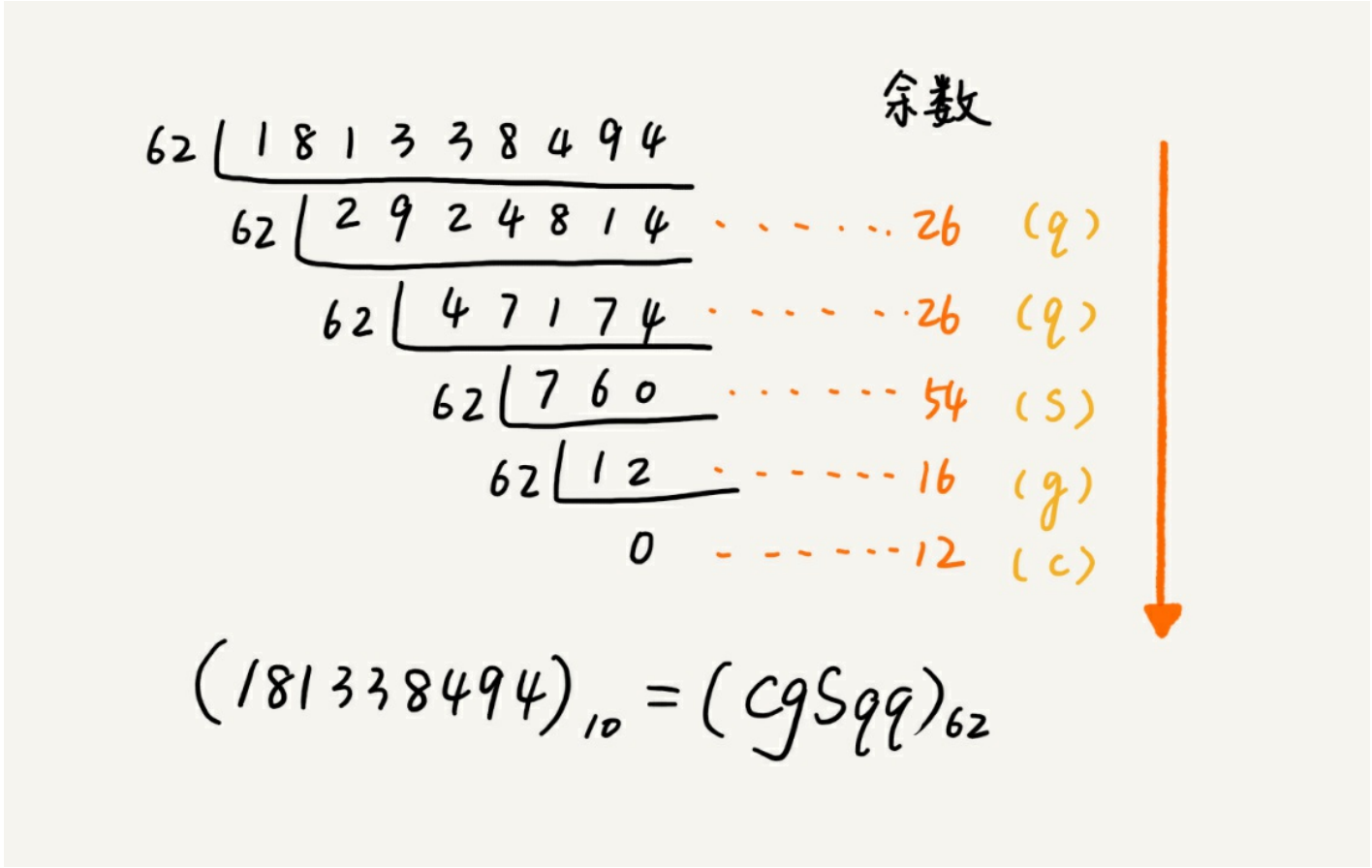
MurmurHash算法提供了两种长度的哈希值，一种是32bits，一种是128bits。为了让最终生成的短网址尽可能短，我们可以选择32bits的哈希值。对于开头那个GitHub网址，经过MurmurHash计算后，得到的哈希值就是181338494。我们再拼上短网址服务的域名，就变成了最终的短网址http://t.cn/181338494（其中，http://t.cn 是短网址服务的域名）。

1.如何让短网址更短？

不过，你可能已经看出来了，通过MurmurHash算法得到的短网址还是很长啊，而且跟我们开头那个网址的格式好像也不一样。别着急，我们只需要稍微改变一个哈希值的表示方法，就可以轻松把短网址变得更短些。

我们可以将10进制的哈希值，转化成更高进制的哈希值，这样哈希值就变短了。我们知道，16进制中，我们用A～E，来表示10～15。在网址URL中，常用的合法字符有0～9、a～z、A～Z这样62个字符。为了让哈希值表示起来尽可能短，我们可以将

10进制的哈希值转化成62进制。具体的计算过程，我写在这里了。最终用62进制表示的短网址就是http://t.cn/cgSqq。



2.如何解决哈希冲突问题?

不过，我们前面讲过，哈希算法无法避免的一个问题，就是哈希冲突。尽管MurmurHash算法，冲突的概率非常低。但是，一旦冲突，就会导致两个原始网址被转化成同一个短网址。当用户访问短网址的时候，我们就无从判断，用户想要访问的是哪一个原始网址了。这个问题该如何解决呢?

一般情况下，我们会保存短网址跟原始网址之间的对应关系，以便后续用户在访问短网址的时候，可以根据对应关系，查找到原始网址。存储这种对应关系的方式有很多，比如我们自己设计存储系统或者利用现成的数据库。前面我们讲到的数据库有MySQL、Redis。我们就拿MySQL来举例。假设短网址与原始网址之间的对应关系，就存储在MySQL数据库中。

当有一个新的原始网址需要生成短网址的时候，我们先利用MurmurHash算法，生成短网址。然后，我们拿这个新生成的短网址，在MySQL数据库中查找。

如果没有找到相同的短网址，这也就表明，这个新生成的短网址没有冲突。于是我们就将这个短网址返回给用户（请求生成短网址的用户），然后将这个短网址与原始网址之间的对应关系，存储到MySQL数据库中。

如果我们在数据库中，找到了相同的短网址，那也并不一定说明就冲突了。我们从数据库中，将这个短网址对应的原始网址也取出来。如果数据库中的原始网址，跟我们现在正在处理的原始网址是一样的，这就说明已经有人请求过这个原始网址的短网址了。我们就可以拿这个短网址直接用。如果数据库中记录的原始网址，跟我们正在处理的原始网址不一样，那就说明哈希算法发生了冲突。不同的原始网址，经过计算，得到的短网址重复了。这个时候，我们该怎么办呢?

我们可以给原始网址拼接一串特殊字符，比如“[DUPLICATED]”，然后跟再重新计算哈希值，两次哈希计算都冲突的概率，显然是非常低的。假设出现非常极端的情况，又发生冲突了，我们可以再换一个拼接字符串，比如“[OHMYGOD]”，再计算哈希值。然后把计算得到的哈希值，跟原始网址拼接了特殊字符串之后的文本，一并存储在MySQL数据库中。

当用户访问短网址的时候，短网址服务先通过短网址，在数据库中查找到对应的原始网址。如果原始网址有拼接特殊字符（这个很容易通过字符串匹配算法找到），我们就先将特殊字符去掉，然后再将不包含特殊字符的原始网址返回给浏览器。

3.如何优化哈希算法生成短网址的性能?

为了判断生成的短网址是否冲突，我们需要拿生成的短网址，在数据库中查找。如果数据库中存储的数据非常多，那查找起来就会非常慢，势必影响短网址服务的性能。那有没有什么优化的手段呢?

还记得我们之前讲的MySQL数据库索引吗？我们可以给短网址字段添加B+树索引。这样通过短网址查询原始网址的速度就提高了很多。实际上，在真实的软件开发中，我们还可以通过一个小技巧，来进一步提高速度。

在短网址生成的过程中，我们会跟数据库打两次交道，也就是会执行两条SQL语句。第一个SQL语句是通过短网址查询短网址与原始网址的对应关系，第二个SQL语句是将新生成的短网址和原始网址之间的对应关系存储到数据库。

我们知道，一般情况下，数据库和应用服务（只做计算不存储数据的业务逻辑部分）会部署在两个独立的服务器或者虚拟服务器上。那两条SQL语句的执行就需要两次网络通信。这种IO通信耗时以及SQL语句的执行，才是整个短网址服务的性能瓶颈所在。所以，为了提高性能，我们需要尽量减少SQL语句。那又该如何减少SQL语句呢?

我们可以给数据库中的短网址字段，添加一个唯一索引（不止是索引，还要求表中不能有重复的数据）。当有新的原始网址需要生成短网址的时候，我们并不会先拿生成的短网址，在数据库中查找判重，而是直接将生成的短网址与对应的原始网址，尝试存储到数据库中。如果数据库能够将数据正常写入，那说明并没有违反唯一索引，也就是说，这个新生成的短网址并没有冲突。

当然，如果数据库反馈违反唯一性索引异常，那我们还得重新执行刚刚讲过的“查询、写入”过程，SQL语句执行的次数不减反增。但是，在大部分情况下，我们把新生成的短网址和对应的原始网址，插入到数据库的时候，并不会出现冲突。所以，大部分情况下，我们只需要执行一条写入的SQL语句就可以了。所以，从整体上看，总的SQL语句执行次数会大大减少。

实际上，我们还有另外一个优化SQL语句次数的方法，那就是借助布隆过滤器。

我们把已经生成的短网址，构建成布隆过滤器。我们知道，布隆过滤器是比较节省内存的一种存储结构，长度是10亿的布隆过滤器，也只需要125MB左右的内存空间。

当有新的短网址生成的时候，我们先拿这个新生成的短网址，在布隆过滤器中查找。如果查找的结果是不存在，那就说明这个新生成的短网址并没有冲突。这个时候，我们只需要再执行写入短网址和对应原始网页的SQL语句就可以了。通过先查询布隆过滤器，总的SQL语句的执行次数减少了。

到此，利用哈希算法来生成短网址的思路，我就讲完了。实际上，这种解决思路已经完全满足需求了，我们已经可以直接用到真实的软件开发中。不过，我们还有另外一种短网址的生成算法，那就是利用自增的ID生成器来生成短网址。我们接下来就看一下，这种算法是如何工作的？对于哈希算法生成短网址来说，它又有什么优势和劣势?

如何通过ID生成器生成短网址?

我们可以维护一个ID自增生成器。它可以生成1、2、3…这样自增的整数ID。当短网址服务接收到一个原始网址转化成短网址的请求之后，它先从ID生成器中取一个号码，然后将其转化成62进制表示法，拼接到短网址服务的域名（比如http://t.cn/）后面，就形成了最终的短网址。最后，我们还是会把生成的短网址和对应的原始网址存储到数据库中。

理论非常简单好理解。不过，这里有几个细节问题需要处理。

1.相同的原始网址可能会对应不同的短网址

每次新来一个原始网址，我们就生成一个新的短网址，这种做法就会导致两个相同的原始网址生成了不同的短网址。这个该如何处理呢？实际上，我们有两种处理思路。

第一种处理思路是**不做处理**。听起来有点无厘头，我稍微解释下你就明白了。实际上，相同的原始网址对应不同的短网址，这个用户是可以接受的。在大部分短网址的应用场景里，用户只关心短网址能否正确地跳转到原始网址。至于短网址长什么样子，他其实根本就不关心。所以，即便是同一个原始网址，两次生成的短网址不一样，也并不会影响到用户的使用。

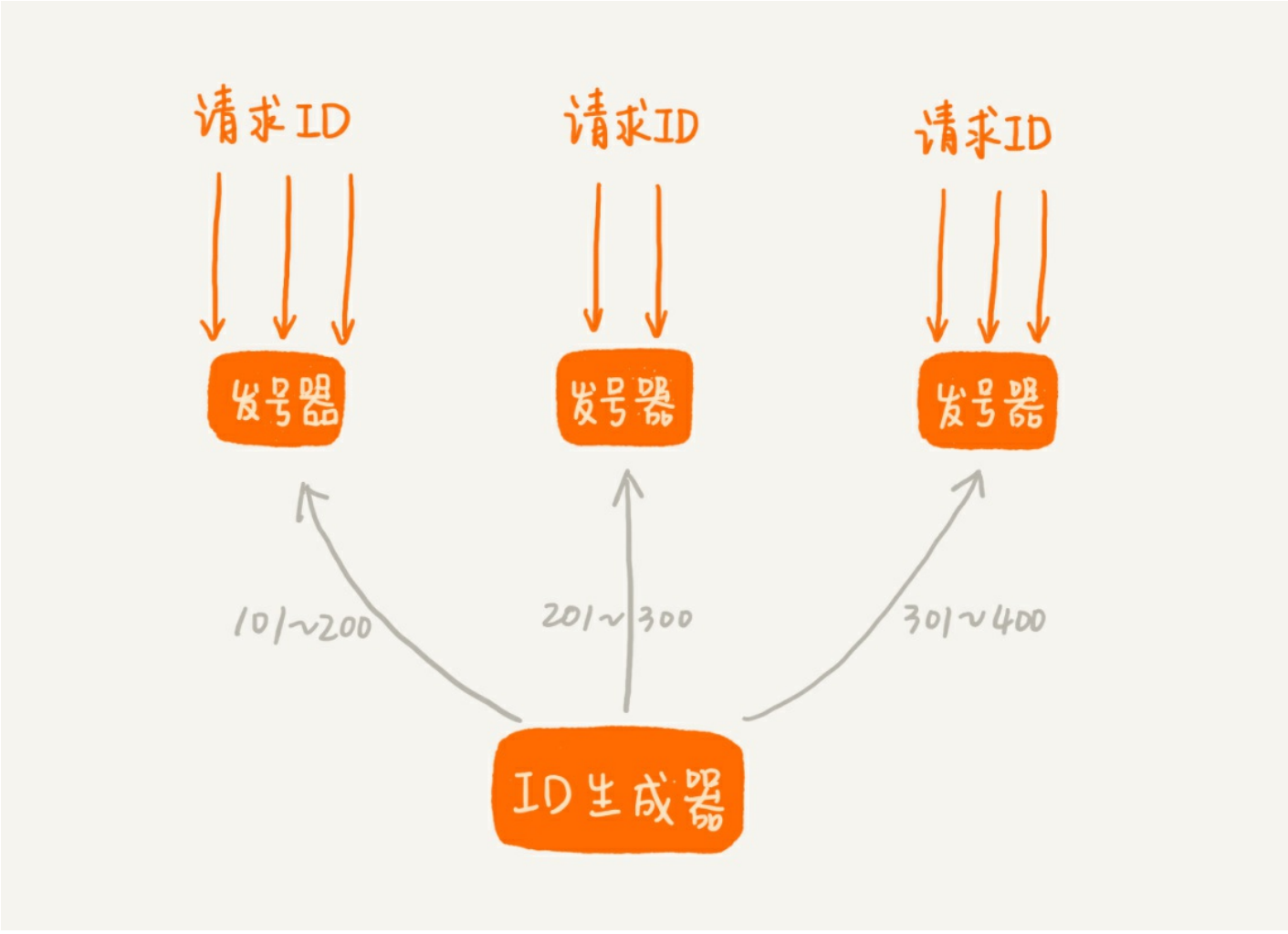
第二种处理思路是**借助哈希算法生成短网址的处理思想，**当要给一个原始网址生成短网址的时候，我们要先拿原始网址在数据库中查找，看数据库中是否已经存在相同的原始网址了。如果数据库中存在，那我们就取出对应的短网址，直接返回给用户。

不过，这种处理思路有个问题，我们需要给数据库中的短网址和原始网址这两个字段，都添加索引。短网址上加索引是为了提高用户查询短网址对应的原始网页的速度，原始网址上加索引是为了加快刚刚讲的通过原始网址查询短网址的速度。这种解决思路虽然能满足“相同原始网址对应相同短网址”这样一个需求，但是是有代价的：一方面两个索引会占用更多的存储空间，另一方面索引还会导致插入、删除等操作性能的下降。

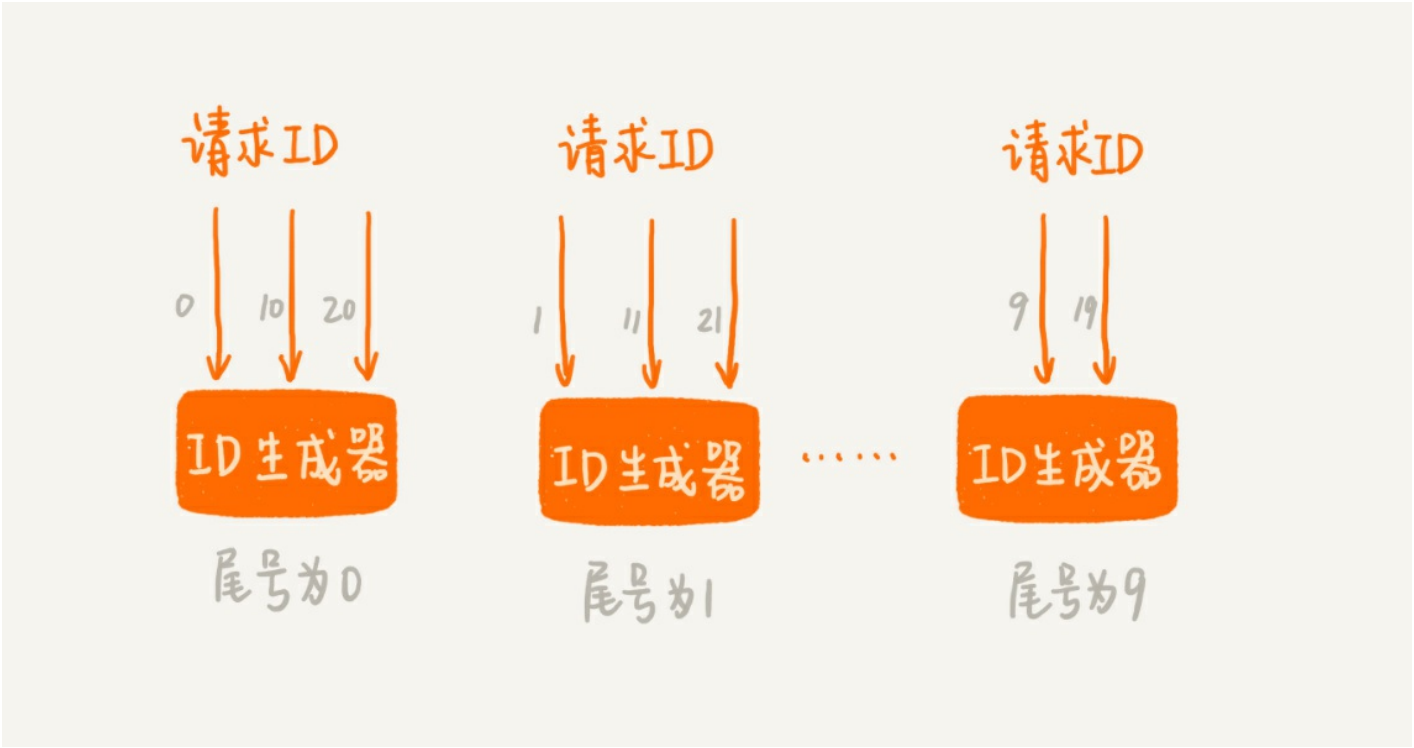
2.如何实现高性能的ID生成器?

实现ID生成器的方法有很多，比如利用数据库自增字段。当然我们也可以自己维护一个计数器，不停地加一加一。但是，一个计数器来应对频繁的短网址生成请求，显然是有点吃力的（因为计数器必须保证生成的ID不重复，笼统概念上讲，就是需要加锁）。如何提高ID生成器的性能呢？关于这个问题，实际上，有很多解决思路。我这里给出两种思路。

第一种思路是借助第54节中讲的方法。我们可以给ID生成器装多个前置发号器。我们批量地给每个前置发号器发送ID号码。当我们接受到短网址生成请求的时候，就选择一个前置发号器来取号码。这样通过多个前置发号器，明显提高了并发发号的能力。



第二种思路跟第一种差不多。不过，我们不再使用一个ID生成器和多个前置发号器这样的架构，而是，直接实现多个ID生成器同时服务。为了保证每个ID生成器生成的ID不重复。我们要求每个ID生成器按照一定的规则，来生成ID号码。比如，第一个ID生成器只能生成尾号为0的，第二个只能生成尾号为1的，以此类推。这样通过多个ID生成器同时工作，也提高了ID生成的效率。



总结引申

今天，我们讲了短网址服务的两种实现方法。我现在来稍微总结一下。

第一种实现思路是通过哈希算法生成短网址。我们采用计算速度快、冲突概率小的MurmurHash算法，并将计算得到的10进制数，转化成62进制表示法，进一步缩短短网址的长度。对于哈希算法的哈希冲突问题，我们通过给原始网址添加特殊前缀字符，重新计算哈希值的方法来解决。

第二种实现思路是通过ID生成器来生成短网址。我们维护一个ID自增的ID生成器，给每个原始网址分配一个ID号码，并且同样转成62进制表示法，拼接到短网址服务的域名之后，形成最终的短网址。

课后思考

1. 如果我们还要额外支持用户自定义短网址功能（http//t.cn/{用户自定部分}），我们又该如何改造刚刚的算法呢？
2. 我们在讲通过ID生成器生成短网址这种实现思路的时候，讲到相同的原始网址可能会对应不同的短网址。针对这个问题，其中一个解决思路就是，不做处理。但是，如果每个请求都生成一个短网址，并且存储在数据库中，那这样会不会撑爆数据库呢？我们又该如何解决呢？

今天是农历的大年三十，我们专栏的正文到这里也就全部结束了。从明天开始，我会每天发布一篇练习题，内容针对专栏涉及的数据结构和算法。从初一到初七，帮你复习巩固所学知识，拿下数据结构和算法，打响新年进步的第一枪！明天见！

精选留言

Smallfly

随着新年的到来，我们的算法专栏也到了尾声。有点怀念那段时间工作不忙，一天能有好几个小时，阅读和思考算法专栏。

专栏给我带来的收获不仅仅是数据结构和算法的知识。在这之前虽然也每天学习，但总是东一块西一块，没有系统和脉络，一段时间之后，看似学了很多，但并没有什么效果。

在学习算法课程的过程中，基本都在学习和思考专栏的内容。一般第一天过一遍概念，一边阅读一边手敲。第二天会把重点放在思考上，为什么需要这种数据结构和算法，它的利弊是什么，以及解答思考题。

真正有收获的是思考和实践而不是阅读，阅读只是表象；如果只是阅读，也没有生字，会很轻松，但效果甚微。

这让我切身体会到系统和专注的重要性。

这种经历帮助我在实际工作中解决不少问题，不是说用到了哪种算法，而是在遇到问题时善于花时间去思考解决方案，而不是一味地寻找替代方案，把问题绕过去。

紧跟着老师的脚步学下来了，不能说都掌握了，但有几点学习的心得想跟大家分享。

1、迈出第一步。很多事情不是我们不会做，而是不想开始。一想到前面可能会有重重困难，思想上就先败下阵来了。只要开始一点点的弄懂，重在培养兴趣。

2、慢一点，不贪多。可能有同学落下很多课，看看还有这么多没学，干脆就放弃了。其实并不需要掌握所有的，可以挑选自己感兴趣的内容，细嚼慢咽，先掌握一部分。再从已知探索未知，其实文章之间很多是有关联的，学习过程中可能自个儿就串起来了。

3、多动手。阅读和思考还是远远不够的，带着文章的思路，用自己熟悉的语言实现一遍，跑起来测一测，会更有成就感，也

是自己学习的痕迹。

虽然专栏学完了，然而有些内容很快就会忘记，后面我还会偶尔拿出来读一读，回顾一下思路。我现在已经欣然接受，学习的东西很快会遗忘，但回顾的时候会迅速地想起来，印象也更深刻了。

最后，祝王争老师和编辑小姐姐新年快乐，辛苦了。(´▽`ʃ♡ƪ)

2019-02-04 00:41

微秒

坚持到了最后，虽然只看不写，但也加深了对数据结构的认识，接下来刷第二遍的时候再加深代码实践。最后祝大家新年快乐！顺便说句，老师的这个专栏真的很良心，谢谢了！

2019-02-04 14:11

李

以前觉得数据结构和算法很难，学了之后，确实也难，但通过系统学习，心中有了一张完整的地图，以后只要不断反复看，反复学习，反复练习，一定能真正融合贯通。
另外，最大的感受是学了数据结构和算法后，看其它中间件和框架的源代码，发现大部分底层就是数据结构和算法。感觉练了九阳神功一样，学习其它功夫快了很多

2019-02-04 09:13

纯洁的憎恶

1.短网址的自定义部分是要展示给用户的。是否可以把自定义部分作为第三个字段存入数据库。如果不同用户对相同原网址申请短网址自定义部分不同。要么不允许这种行为，否则就得把自定义部分与原网址拼接输入哈希函数，以实现区分。

2019-02-07 20:07

一涛

1. 首先查询“用户自定义部分”是否与已经生成的短网址冲突，如果冲突，只能提示用户进行修改。如果不冲突，将“用户自定义部分”和对应的原始网址写入数据库即可。

2. 给原始网址加唯一索引。如果写入异常，说明原始网址已经存在，再根据原始网址查询一次，取出短网址返回给用户。

不知道回答对不对，请老师指正

2019-02-04 11:10

吴...

兴趣是最好的老师，这话没有错。如果没有兴趣那就去找。就我个人看法，很多时候一开始不一定非要搞那么枯燥的东西，做一些有趣的东西，慢慢培养自己的兴趣，自己就会有好奇心去往深处学习，如果以来就弄那些很“艰深”的东西，可能不能坚持多久，“从入门到放弃”。学习老师这个专栏，我最大的收获就在于老师把平时课上那些算法讲活了，应用到具体场景之中，相较于一些为了练习而联系的习题，这让我更能体会到算法之美。也在不断激发我的好奇心。我希望自己能够一直抱持这份好奇心。继续努力学习。

2019-02-04 10:46

想当上帝的司机

用户自定义的，可以将用户的id拼接到hash前的网址上

2019-02-04 07:26

实验室清洁工

应该是16进制吧，62进制？？？

2019-02-13 08:28

bens

id生成器用snowflake算法解决

2019-02-11 14:20

Sharry

问题一:

- 尝试将用户 自定义后的短网址 和 原网址的映射关系 存入数据库
- 插入成功, 则提示用户短网址生成成功
- 若插入失败, 说明存在冲突, 则进行判重处理
- 若数据库中短网址对应的原网址与当前正在处理的相同, 提示该短网址有效
- 若数据库中短网址对应的原网址与当前正在处理的不相同, 提示该短网址已被占用

问题二:
可以使用布隆过滤器进行判重验证, 通过之后再插入
2019-02-11 11:35


小美
王老师短网址有什么作用吗？ 我上网查了下理由不能说服我?
可能网上说得比较浅显，王老师方便指导下吗?
2019-02-09 22:23


纯洁的憎恶
2.无论是给原网址建立唯一索引，还是采用布隆过滤器，都无法回避长短网址两个字段均加索引，这样会增加数据库维护的复杂度。单独建立原始网址的散列表，存储其在数据库中的位置。这样当发现冲突时，可以快速查询原网址在数据库中的位置，并返回短网址。只是在将新的对应关系写入数据库时，也要将数据库位置同步写入原网址散列表。并且当数据库变更时要保持原网址散列表的一致性。这样会比两字段都加索引便捷么?
2019-02-07 19:48


纯洁的憎恶
通过哈希函数，在长网址字符串基础上，生成短网址哈希值。将哈希值从10进制提升至62进制，进一步缩短短网址长度。为了通过短网址回溯到原网址，需要建立长短网址的对应关系，存入数据库。

为了避免散列冲突，需要在在建立新的对应关系时，查询数据库中是否已有短网址，若有再检查长网址是否一致，若不一致则发生冲突，需要给新的长网址字符串加前缀，再用哈希函数生成短网址，直到没有冲突，最终将前缀、对应关系均存入数据库。

为方便查询，需要在数据库中建立短网址的索引（B+树）。减少SQL语句数量也可以提高性能，把查询+写入两条语句，简化为写入一条。代价是设置短网址唯一索引，不允许出现重复，这样当重复写入时数据库才会报错，此时再通过查询、前缀、写入的方式解决散列冲突。也可以针对短网址建立布隆过滤器，当新的短网址不在过滤器中则正常写入，否则通过查询判重并解决冲突。

另一种方式是通过全局计数器，给每个请求的原网址分配一个序号，作为短网址的主要部分。但它可能造成同一个原网址对应多个短网址的现象（虽然不影响应用体验）。为提高给号的并发性能，可以针对不同号段设置多个发号器并行发号。
2019-02-07 19:48


纯洁的憎恶
期待思考题答案!
2019-02-07 12:56


纯洁的憎恶
坚持下来啦！收获满满！感谢王争老师!
2019-02-07 12:53


梦里
老师辛苦！新年快乐!
2019-02-04 21:05


行者
针对问题2，可以使用布隆过滤器来处理，如果url已经生成过短网址，不做处理。
2019-02-04 17:46


与非
最后一课在一年的最后一天结束，这也算辞旧迎新了吧～希望老师能在最后能出个课后思考题的总结～
2019-02-04 15:36


吴天
很有收获 祝大家新年快乐
2019-02-04 14:58