

我必须得告诉大家的 MySQL 优化原理

转载 高效运维 最后发布于2018-05-08 13:52:59 阅读数 1115 ☆ 收藏

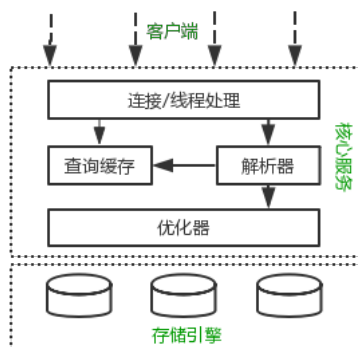
点击蓝字，轻松关注



说起MySQL的查询优化，相信大家收藏了一堆奇技淫巧：不能使用SELECT *、不使用NULL字段、合理创建索引、为字段选择合适的数据类型..... 你是否真了解这些优化技巧？是否理解其背后的工作原理？在实际场景下性能真有提升吗？我想未必。因而理解这些优化建议背后的原理就尤为重要，希望本文能让你重视这些优化建议，并在实际业务场景下合理的运用。

MySQL逻辑架构

如果能在头脑中构建一幅MySQL各组件之间如何协同工作的架构图，有助于深入理解MySQL服务器。下图展示了MySQL的逻辑架构图。



MySQL逻辑架构，来自：高性能MySQL

MySQL逻辑架构整体分为三层，最上层为客户端层，并非MySQL所独有，诸如：连接处理、授权认证、安全等功能均在这一层处理。

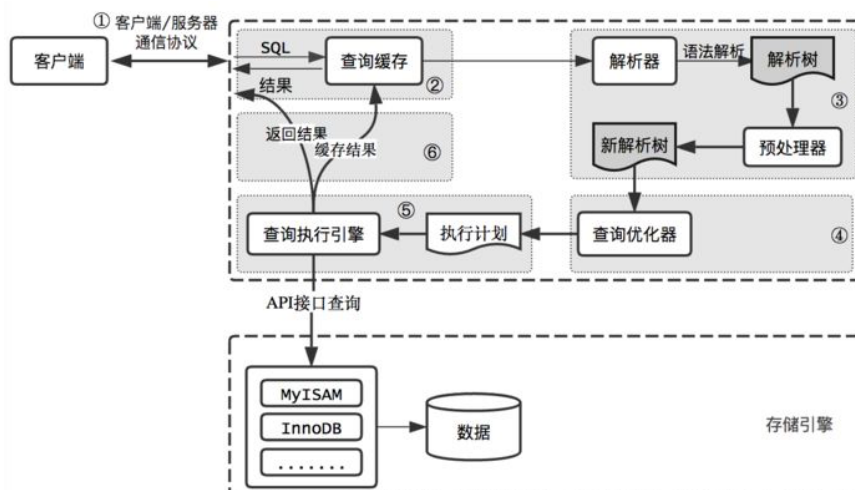
MySQL大多数核心服务均在中间这一层，包括查询解析、分析、优化、缓存、内置函数(比如：时间、数学、加密等函数)。所有的跨存储引擎的功能也在这里：存储过程、触发器、视图等。

最下层为存储引擎，其负责MySQL中的数据存储和提取。和Linux下的文件系统类似，每种存储引擎都有其优势和劣势。中间的服务层通过API与存储引擎通过这些API接口屏蔽了不同存储引擎间的差异。

MySQL查询过程

我们总是希望MySQL能够获得更高的查询性能，最好的办法是弄清楚MySQL是如何优化和执行查询的。一旦理解了这一点，就会发现：很多的查询优化工作就是遵循一些原则让MySQL的优化器能够按照预想的合理方式运行而已。

当向MySQL发送一个请求的时候，MySQL到底做了些什么呢？



MySQL查询过程

客户端/服务端通信协议

MySQL客户端/服务端通信协议是“半双工”的：在任一时刻，要么是服务器向客户端发送数据，要么是客户端向服务器发送数据，这两个动作不能同时发生。端开始发送消息，另一端要接收完整个消息才能响应它，所以我们无法也无须将一个消息切成小块独立发送，也没有办法进行流量控制。

客户端用一个单独的数据包将查询请求发送给服务器，所以当查询语句很长的时候，需要设置max_allowed_packet参数。但是需要注意的是，如果查询实在大，服务端会拒绝接收更多数据并抛出异常。

与之相反的是，服务器响应给用户的数据通常会很多，由多个数据包组成。但是当服务器响应客户端请求时，客户端必须完整的接收整个返回结果，而不能只取前面几条结果，然后让服务器停止发送。因而在实际开发中，尽量保持查询简单且只返回必需的数据，减小通信间数据包的大小和数量是一个非常好的习惯。是查询中尽量避免使用SELECT *以及加上LIMIT限制的原因之一。

查询缓存

在解析一个查询语句前，如果查询缓存是打开的，那么MySQL会检查这个查询语句是否命中查询缓存中的数据。如果当前查询恰好命中查询缓存，在检查一权限后直接返回缓存中的结果。这种情况下，查询不会被解析，也不会生成执行计划，更不会执行。

MySQL将缓存存放在一个引用表（不要理解成table，可以认为是类似于HashMap的数据结构），通过一个哈希值索引，这个哈希值通过查询本身、当前要查询的数据库、客户端协议版本号等一些可能影响结果的信息计算得来。所以两个查询在任何字符上的不同（例如：空格、注释），都会导致缓存不会命中。

如果查询中包含任何用户自定义函数、存储函数、用户变量、临时表、mysql库中的系统表，其查询结果都不会被缓存。比如函数NOW()或者CURRENT_DATE()会因为不同的查询时间，返回不同的查询结果，再比如包含CURRENT_USER或者CONNECTION_ID的查询语句会因为不同的用户而返回不同的结果，将这样的查询结果缓存起来没有任何的意义。

既然是缓存，就会失效，那查询缓存何时失效呢？MySQL的查询缓存系统会跟踪查询中涉及的每个表，如果这些表（数据或结构）发生变化，那么和这张表所有缓存数据都将失效。正因为如此，在任何的写操作时，MySQL必须将对对应表的所有缓存都设置为失效。如果查询缓存非常大或者碎片很多，这个操作就会带来很大的系统消耗，甚至导致系统僵死一会儿。而且查询缓存对系统的额外消耗也不仅仅在写操作，读操作也不例外：

任何的查询语句在开始之前都必须经过检查，即使这条SQL语句永远不会命中缓存

如果查询结果可以被缓存，那么执行完成后，会将结果存入缓存，也会带来额外的系统消耗

基于此，我们要知道并不是什么情况下查询缓存都会提高系统性能，缓存和失效都会带来额外消耗，只有当缓存带来的资源节约大于其本身消耗的资源时，才会给系统带来性能提升。但要如何评估打开缓存是否能够带来性能提升是一件非常困难的事情，也不在本文讨论的范畴内。如果系统确实存在一些性能问题，可以尝试打开查询缓存，并在数据库设计上做一些优化，比如：

用多个小表代替一个大表，注意不要过度设计

批量插入代替循环单条插入

合理控制缓存空间大小，一般来说其大小设置为几十兆比较合适

可以通过SQL_CACHE和SQL_NO_CACHE来控制某个查询语句是否需要进行缓存

最后的忠告是不要轻易打开查询缓存，特别是写密集型应用。如果你实在是忍不住，可以将query_cache_type设置为DEMAND，这时只有加入SQL_CACHE才会走缓存，其他查询则不会，这样可以非常自由地控制哪些查询需要被缓存。

当然查询缓存系统本身是非常复杂的，这里讨论的也只是很小的一部分，其他更深入的话题，比如：缓存是如何使用内存的？如何控制内存的碎片化？事务对缓存有何影响等等，读者可以自行阅读相关资料，这里权当抛砖引玉吧。

语法规则和预处理

MySQL通过关键字将SQL语句进行解析，并生成一颗对应的解析树。这个过程解析器主要通过语法规则来验证和解析。比如SQL中是否使用了错误的关键字的顺序是否正确等等。预处理则会根据MySQL规则进一步检查解析树是否合法。比如检查要查询的数据表和数据列是否存在等等。

查询优化

经过前面的步骤生成的语法树被认为是合法的了，并且由优化器将其转化成查询计划。多数情况下，一条查询可以有很多种执行方式，最后都返回相应的结果。优化器的作用就是找到这其中最好的执行计划。

MySQL使用基于成本的优化器，它尝试预测一个查询使用某种执行计划时的成本，并选择其中成本最小的一个。在MySQL可以通过查询当前会话的last_query_cost的值来得到其计算当前查询的成本。

```
mysql> select * from t_message limit 10;
...省略结果集

mysql> show status like 'last_query_cost';
+-----+-----+
| Variable_name | Value          |
+-----+-----+
| Last_query_cost | 6391.799000    |
+-----+-----+
```

示例中的结果表示优化器认为大概需要做6391个数据页的随机查找才能完成上面的查询。这个结果是根据一些列的统计信息计算得来的，这些统计信息包括表或者索引的页面个数、索引的基数、索引和数据行的长度、索引的分布情况等等。

有非常多的原因会导致MySQL选择错误的执行计划，比如统计信息不准确、不会考虑不受其控制的操作成本（用户自定义函数、存储过程）、MySQL认为的我们想的不一樣（我们希望执行时间尽可能短，但MySQL值选择它认为成本小的，但成本小并不意味着执行时间短）等等。

MySQL的查询优化器是一个非常复杂的部件，它使用了非常多的优化策略来生成一个最优的执行计划：

- 重新定义表的关联顺序（多张表关联查询时，并不一定按照SQL中指定的顺序进行，但有一些技巧可以指定关联顺序）

- 优化MIN()和MAX()函数（找某列的最小值，如果该列有索引，只需要查找B+Tree索引最左端，反之则可以找到最大值，具体原理见下文）

- 提前终止查询（比如：使用Limit时，查找到满足数量的结果集后会立即终止查询）

- 优化排序（在老版本MySQL会使用两次传输排序，即先读取行指针和需要排序的字段在内存中对其排序，然后再根据排序结果去读取数据行，而新版本是单次传输排序，也就是一次读取所有的数据行，然后根据给定的列排序。对于I/O密集型应用，效率会高很多）

随着MySQL的不断发展，优化器使用的优化策略也在不断的进化，这里仅仅介绍几个非常常用且容易理解的优化策略，其他的优化策略，大家自行查阅吧。

查询执行引擎

在完成解析和优化阶段以后，MySQL会生成对应的执行计划，查询执行引擎根据执行计划给出的指令逐步执行得出结果。整个执行过程的大部分操作均是通存储引擎实现的接口来完成，这些接口被称为handler API。查询过程中的每一张表由一个handler实例表示。实际上，MySQL在查询优化阶段就为每一张表生成一个handler实例，优化器可以根据这些实例的接口来获取表的相关信息，包括表的所有列名、索引统计信息等。存储引擎接口提供了非常丰富的功能，但其底几十个接口，这些接口像搭积木一样完成了一次查询的大部分操作。

返回结果给客户端

查询执行的最后一个阶段就是将结果返回给客户端。即使查询不到数据，MySQL仍然会返回这个查询的相关信息，比如该查询影响到的行数以及执行时间等。如果查询缓存被打开且这个查询可以被缓存，MySQL也会将结果存放到缓存中。

结果集返回客户端是一个增量且逐步返回的过程。有可能MySQL在生成第一条结果时，就开始向客户端逐步返回结果集了。这样服务端就无须存储太多结果过多内存，也可以让客户第一时间获得返回结果。需要注意的是，结果集中的每一行都会以一个满足①中所描述的通信协议的数据包发送，再通过TCP协议传输，在传输过程中，可能对MySQL的数据包进行缓存然后批量发送。

回头总结一下MySQL整个查询执行过程，总的来说分为6个步骤：

- 客户端向MySQL服务器发送一条查询请求

- 服务器首先检查查询缓存，如果命中缓存，则立刻返回存储在缓存中的结果。否则进入下一阶段

- 服务器进行SQL解析、预处理、再由优化器生成对应的执行计划

- MySQL根据执行计划，调用存储引擎的API来执行查询

- 将结果返回给客户端，同时缓存查询结果

性能优化建议

看了这么多，你可能会期待给出一些优化手段，是的，下面会从3个不同方面给出一些优化建议。但请等等，还有一句忠告要先送给你：**不要听信你看到的关于“绝对真理”**，包括本文所讨论的内容，而应该是在实际的业务场景下通过测试来验证你关于执行计划以及响应时间的假设。

Scheme设计与数据类型优化

选择数据类型只要遵循小而简单的原则就好，越小的数据类型通常会更快，占用更少的磁盘、内存，处理时需要的CPU周期也更少。越简单的数据类型在计算需要更少的CPU周期，比如，整型就比字符操作代价低，因而会使用整型来存储ip地址，使用DATETIME来存储时间，而不是使用字符串。

这里总结几个可能容易理解错误的技巧：

通常来说把可为NULL的列改为NOT NULL不会对性能提升有多少帮助，只是如果计划在列上创建索引，就应该将该列设置为NOT NULL。

对整数类型指定宽度，比如INT(11)，没有任何卵用。INT使用32位（4个字节）存储空间，那么它的表示范围已经确定，所以INT(1)和INT(20)对于存储是相同的。

UNSIGNED表示不允许负值，大致可以使正数的上限提高一倍。比如TINYINT存储范围是-128 ~ 127，而UNSIGNED TINYINT存储的范围却是0 - 255。

通常来讲，没有太大的必要使用DECIMAL数据类型。即使是在需要存储财务数据时，仍然可以使用BIGINT。比如需要精确到万分之一，那么可以将数一百万然后使用BIGINT存储。这样可以避免浮点数计算不准确和DECIMAL精确计算代价高的问题。

TIMESTAMP使用4个字节存储空间，DATETIME使用8个字节存储空间。因而，TIMESTAMP只能表示1970 - 2038年，比DATETIME表示的范围小得多且TIMESTAMP的值因时区不同而不同。

大多数情况下没有使用枚举类型的必要，其中一个缺点是枚举的字符串列表是固定的，添加和删除字符串（枚举选项）必须使用ALTER TABLE（如果在列表末尾追加元素，不需要重建表）。

schema的列不要太多。原因是存储引擎的API工作时需要在服务器层和存储引擎层之间通过行缓冲格式拷贝数据，然后在服务器层将缓冲内容解码成各个这个转换过程的代价是非常高的。如果列太多而实际使用的列又很少的话，有可能会造成CPU占用过高。

大表ALTER TABLE非常耗时，MySQL执行大部分修改表结果操作的方法是用新的结构创建一个空表，从旧表中查出所有的数据插入新表，然后再删表。尤其当内存不足而表又很大，而且还有很大索引的情况下，耗时更久。当然有一些奇技淫巧可以解决这个问题，有兴趣可自行查阅。

创建高性能索引

索引是提高MySQL查询性能的一个重要途径，但过多的索引可能会导致过高的磁盘使用率以及过高的内存占用，从而影响应用程序的整体性能。应当尽量避免才想起添加索引，因为事后可能需要监控大量的SQL才能定位到问题所在，而且添加索引的时间肯定是远大于初始添加索引所需要的时间，可见索引的添加非常有技术含量的。

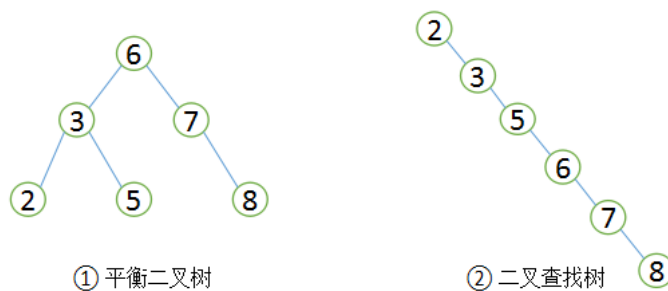
接下来将向你展示一系列创建高性能索引的策略，以及每条策略其背后的工作原理。但在此之前，先了解与索引相关的一些算法和数据结构，将有助于更好地理解内容。

索引相关的数据结构和算法

通常我们所说的索引是指B-Tree索引，它是目前关系型数据库中查找数据最为常用和有效的索引，大多数存储引擎都支持这种索引。使用B-Tree这个术语，MySQL在CREATE TABLE或其它语句中使用了这个关键字，但实际上不同的存储引擎可能使用不同的数据结构，比如InnoDB就是使用的B+Tree。

B+Tree中的B是指balance，意为平衡。需要注意的是，B+树索引并不能找到一个给定键值的具体行，它找到的只是被查找数据行所在的页，接着数据库会读到内存，再在内存中进行查找，最后得到要查找的数据。

在介绍B+Tree前，先了解一下二叉查找树，它是一种经典的数据结构，其左子树的值总是小于根的值，右子树的值总是大于根的值，如下图①。如果要在该查找值为5的记录，其大致流程：先找到根，其值为6，大于5，所以查找左子树，找到3，而5大于3，接着找3的右子树，总共找了3次。同样的方法，如果查8的记录，也需要查找3次。所以二叉查找树的平均查找次数为 $(3 + 3 + 3 + 2 + 2 + 1) / 6 = 2.3$ 次，而顺序查找的话，查找值为2的记录，仅需要1次，但查找1记录则需要6次，所以顺序查找的平均查找次数为： $(1 + 2 + 3 + 4 + 5 + 6) / 6 = 3.3$ 次，因此大多数情况下二叉查找树的平均查找速度比顺序查找要快。

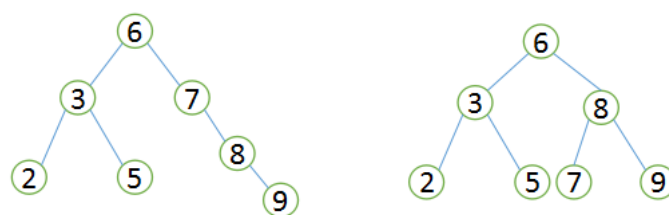


二叉查找树和平衡二叉树

由于二叉查找树可以任意构造，同样的值，可以构造出如图②的二叉查找树，显然这棵二叉树的查询效率和顺序查找差不多。若想二叉查找树的查询性能最好，这棵二叉查找树是平衡的，也即平衡二叉树（AVL树）。

平衡二叉树首先需要符合二叉查找树的定义，其次必须满足任何节点的两个子树的高度差不能大于1。显然图②不满足平衡二叉树的定义，而图①是一棵平衡树。平衡二叉树的查找性能是比较高的（性能最好的是最优二叉树），查询性能越好，维护的成本就越大。比如图①的平衡二叉树，当用户需要插入一个新

节点时，就需要做出如下变动。



插入新值9

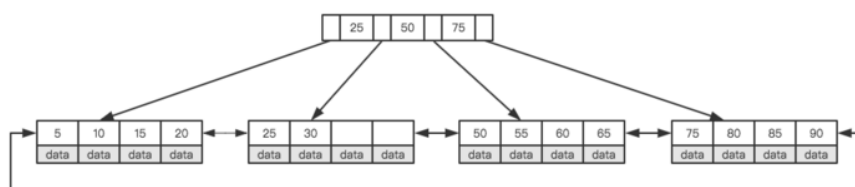
左旋保证平衡

平衡二叉树旋转

通过一次左旋操作就将插入后的树重新变为平衡二叉树是最简单的情况了，实际应用场景中可能需要旋转多次。至此我们可以考虑一个问题，平衡二叉树的实现还不错，实现也非常简单，相应的维护成本还能接受，为什么MySQL索引不直接使用平衡二叉树？

随着数据库中数据的增加，索引本身大小随之增加，不可能全部存储在内存中，因此索引往往以索引文件的形式存储在磁盘上。这样的话，索引查找过程中会产生大量的磁盘I/O消耗，相对于内存存取，I/O存取的消耗要高几个数量级。可以想象一下一棵几百万节点的二叉树的深度是多少？如果将这么大深度的一颗二叉树放磁盘上，每读取一个节点，需要一次磁盘的I/O读取，整个查找的耗时显然是不能够接受的。那么如何减少查找过程中的I/O存取次数？

一种行之有效的解决方法是减少树的深度，将二叉树变为m叉树（多路搜索树），而B+Tree就是一种多路搜索树。理解B+Tree时，只需要理解其最重要的两个特征即可：第一，所有的关键字（可以理解为数据）都存储在叶子节点（Leaf Page），非叶子节点（Index Page）并不存储真正的数据，所有记录节点都是按键顺序存放在同一层叶子节点上。其次，所有的叶子节点由指针连接。如下图为高度为2的简化的B+Tree。



简化B+Tree

怎么理解这两个特征？MySQL将每个节点的大小设置为一个页的整数倍（原因下文会介绍），也就是在节点空间大小一定的情况下，每个节点可以存储更多点，这样每个节点能索引的范围更大更精确。所有的叶子节点使用指针链接的好处是可以进行区间访问，比如上图中，如果查找大于20而小于30的记录，只到节点20，就可以遍历指针依次找到25、30。如果没有链接指针的话，就无法进行区间查找。这也是MySQL使用B+Tree作为索引存储结构的重要原因。

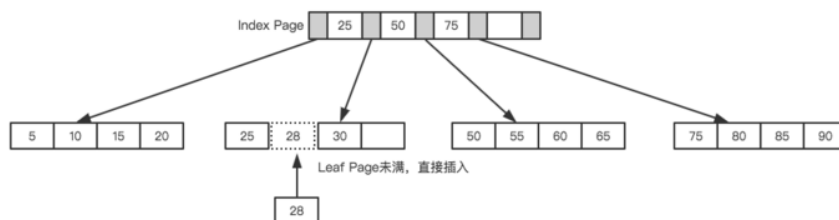
MySQL为何将节点大小设置为页的整数倍，这就需要理解磁盘的存储原理。磁盘本身存取就比主存慢很多，在加上机械运动损耗（特别是普通的机械硬盘）的存取速度往往是主存的几百万分之一，为了尽量减少磁盘I/O，磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定长度的数据放入内存，预读的长度一般为页的整数倍。

页是计算机管理存储器的逻辑块，硬件及OS往往将主存和磁盘存储区分割为连续的大小相等的块，每个存储块称为一页（许多OS中，页的大小通常为4K）。主存和磁盘以页为单位交换数据。当程序要读取的数据不在主存中时，会触发一个缺页异常，此时系统会向磁盘发出读盘信号，磁盘会找到数据的起始位置，然后向后连续读取一页或几页载入内存中，然后一起返回，程序继续运行。

MySQL巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完全载入。为了达到这个目的，每次新建节点时，只申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了读取一个节点只需一次I/O。假设B+Tree的高度为h，一次检索最多需要h-1次I/O（根节点常驻内存），复杂度 $O(h) = O(\log M)$ 。实际应用场景中，M通常较大，常常超过100，因此树的高度一般都比较小，不超过3。

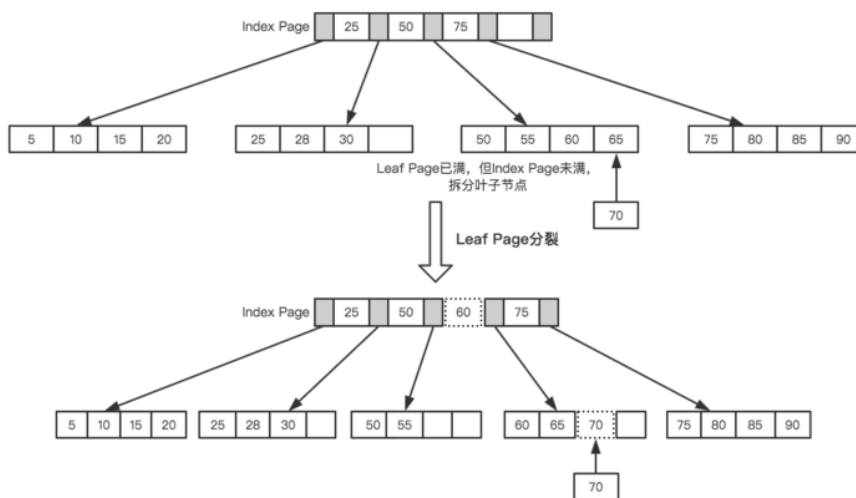
最后简单了解下B+Tree节点的操作，在整体上对索引的维护有一个大概的了解，虽然索引可以大大提高查询效率，但维护索引仍要花费很大的代价，因此合建索引也就尤为重要。

仍以上面的树为例，我们假设每个节点只能存储4个内节点。首先要插入第一个节点28，如下图所示。



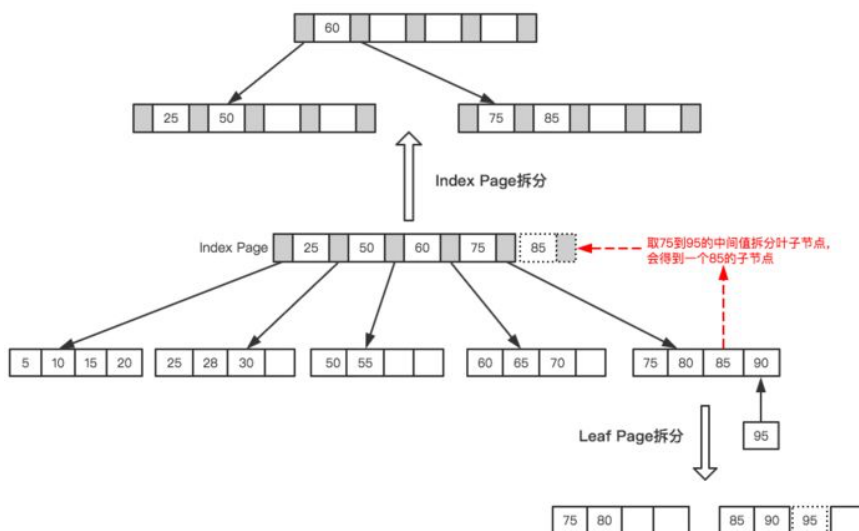
leaf page和index page都没有满

接着插入下一个节点70, 在Index Page中查询后得知应该插入到50 - 70之间的叶子节点, 但叶子节点已满, 这时候就需要进行也分裂的操作, 当前的叶子节点为50, 所以根据中间值来拆分叶子节点, 如下图所示。



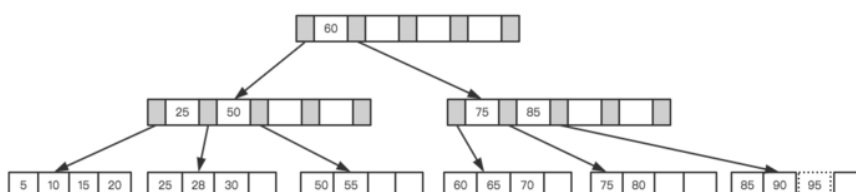
Leaf Page拆分

最后插入一个节点95, 这时候Index Page和Leaf Page都满了, 就需要做两次拆分, 如下图所示。



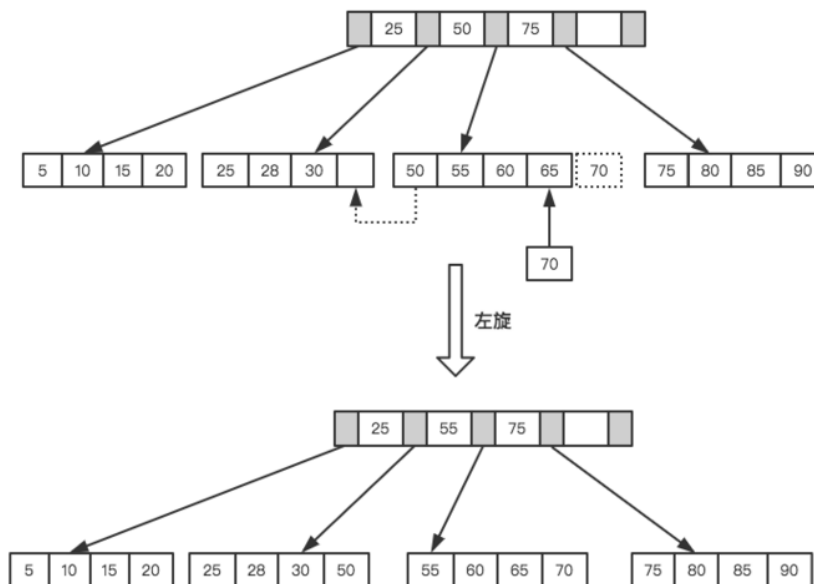
Leaf Page与Index Page拆分

拆分后最终形成了这样一颗树。



最终树

B+Tree为了保持平衡，对于新插入的值需要做大量的拆分页操作，而页的拆分需要I/O操作，为了尽可能的减少页的拆分操作，B+Tree也提供了类似于平衡二叉树旋转功能。当Leaf Page已满但其左右兄弟节点没有满的情况下，B+Tree并不急于去做拆分操作，而是将记录移到当前所在页的兄弟节点上。通常情况下，左旋被先检查用来做旋转操作。就比如上面第二个示例，当插入70的时候，并不会去做页拆分，而是左旋操作。



左旋操作

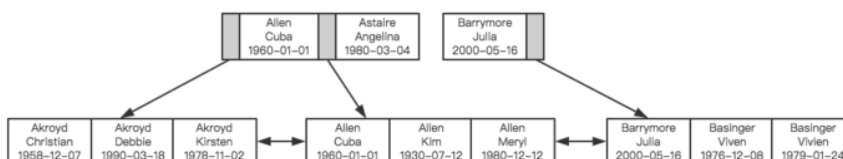
通过旋转操作可以最大限度的减少页分裂，从而减少索引维护过程中的磁盘的I/O操作，也提高索引维护效率。需要注意的是，删除节点跟插入节点类似，仍旋转和拆分操作，这里就不再说明。

高性能策略

通过上文，相信你对B+Tree的数据结构已经有了大致的了解，但MySQL中索引是如何组织数据的存储呢？以一个简单的示例来说明，假如有如下数据表：

```
CREATE TABLE People(  
  last_name varchar(50) not null,  
  first_name varchar(50) not null,  
  dob date not null,  
  gender enum('m','f') not null,  
  key(last_name,first_name,dob)  
);
```

对于表中每一行数据，索引中包含了last_name、first_name、dob列的值，下图展示了索引是如何组织数据存储的。



可以看到，索引首先根据第一个字段来排列顺序，当名字相同时，则根据第三个字段，即出生日期来排序，正是因为这个原因，才有了索引的“最左原则”。

1、MySQL不会使用索引的情况：非独立的列

“独立的列”是指索引列不能是表达式的一部分，也不能是函数的参数。比如：

```
select * from where id + 1 = 5
```

我们很容易看出其等价于 $id = 4$ ，但是MySQL无法自动解析这个表达式，使用函数是同样的道理。

2、前缀索引

如果列很长，通常可以索引开始的部分字符，这样可以有效节约索引空间，从而提高索引效率。

3、多列索引和索引顺序

在多数情况下，在多个列上建立独立的索引并不能提高查询性能。理由非常简单，MySQL不知道选择哪个索引的查询效率更好，所以在老版本，比如MySQL 5.6之前就会随便选择一个列的索引，而新的版本会采用合并索引的策略。举个简单的例子，在一张电影演员表中，在actor_id和film_id两个列上都建立了独立的索引，后有如下查询：

```
select film_id,actor_id from film_actor where actor_id = 1 or film_id = 1
```

老版本的MySQL会随机选择一个索引，但新版本做如下的优化：

```
select film_id,actor_id from film_actor where actor_id = 1
union all
select film_id,actor_id from film_actor where film_id = 1 and actor_id <> 1
```

当出现多个索引做相交操作时（多个AND条件），通常来说一个包含所有相关列的索引要优于多个独立索引。

当出现多个索引做联合操作时（多个OR条件），对结果集的合并、排序等操作需要耗费大量的CPU和内存资源，特别是当其中的某些索引的选择性不高返回合并大量数据时，查询成本更高。所以这种情况下还不如走全表扫描。

因此explain时如果发现有索引合并（Extra字段出现Using union），应该好好检查一下查询和表结构是不是已经是最优的，如果查询和表都没有问题，那只能引建的非常糟糕，应当慎重考虑索引是否合适，有可能一个包含所有相关列的多列索引更适合。

前面我们提到过索引如何组织数据存储的，从图中可以看到多列索引时，索引的顺序对于查询是至关重要的，很明显应该把选择性更高的字段放到索引的前面，通过第一个字段就可以过滤掉大多数不符合条件的数据。

索引选择性是指不重复的索引值和数据表的总记录数的比值，选择性越高查询效率越高，因为选择性越高的索引可以让MySQL在查询时过滤掉更多的行。索引的选择性是1，这是最好的索引选择性，性能也是最好的。

理解索引选择性的概念后，就不难确定哪个字段的选择性较高了，查一下就知道了，比如：

```
SELECT * FROM payment where staff_id = 2 and customer_id = 584
```

是应该创建(staff_id,customer_id)的索引还是应该颠倒一下顺序？执行下面的查询，哪个字段的选择性更接近1就把哪个字段索引前面就好。

```
select count(distinct staff_id)/count(*) as staff_id_selectivity,
       count(distinct customer_id)/count(*) as customer_id_selectivity,
       count(*) from payment
```

多数情况下使用这个原则没有任何问题，但仍然注意你的数据中是否存在一些特殊情况。举个简单的例子，比如要查询某个用户组下有交易的用户信息：

```
select user_id from trade where user_group_id = 1 and trade_amount > 0
```

MySQL为这个查询选择了索引(user_group_id,trade_amount)，如果不考虑特殊情况，这看起来没有任何问题，但实际情况是这张表的大多数数据都是从老系统迁移过来的，由于新老系统的数据不兼容，所以就给老系统迁移过来的数据赋予了一个默认的用户组。这种情况下，通过索引扫描的行数跟全表扫描基本没什么差别，索引也就起不到任何作用。

推广开来说，经验法则和推论在多数情况下是有用的，可以指导我们开发和设计，但实际情况往往会更复杂，实际业务场景下的某些特殊情况可能会摧毁你的设计。

4、避免多个范围条件

实际开发中，我们会经常使用多个范围条件，比如想查询某个时间段内登录过的用户：


```
select user.* from user where login_time > '2017-04-01' and age between 18 and 30;
```

这个查询有一个问题：它有两个范围条件，login_time列和age列，MySQL可以使用login_time列的索引或者age列的索引，但无法同时使用它们。

5、覆盖索引

如果一个索引包含或者说覆盖所有需要查询的字段，那么就没有必要再回表查询，这就称为覆盖索引。覆盖索引是非常有用的工具，可以极大的提高性能。查询只需要扫描索引会带来许多好处：

索引条目远小于数据行大小，如果只读取索引，极大减少数据访问量

索引是有按照列值顺序存储的，对于I/O密集型的范围查询要比随机从磁盘读取每一行数据的IO要少的多

6、使用索引扫描来排序

MySQL有两种方式可以生产有序的结果集，其一是对结果集进行排序的操作，其二是按照索引顺序扫描得出的结果自然是有序的。如果explain的结果中type为index表示使用了索引扫描来做排序。

扫描索引本身很快，因为只需要从一条索引记录移动到相邻的下一条记录。但如果索引本身不能覆盖所有需要查询的列，那么就不得不每扫描一条索引记录回表查询一次对应的行。这个读取操作基本上是随机I/O，因此按照索引顺序读取数据的速度通常要比顺序地全表扫描要慢。

在设计索引时，如果一个索引既能够满足排序，又满足查询，是最好的。

只有当索引的列顺序和ORDER BY子句的顺序完全一致，并且所有列的排序方向也一样时，才能够使用索引来对结果做排序。如果查询需要关联多张表，则有ORDER BY子句引用的字段全部为第一张表时，才能使用索引做排序。ORDER BY子句和查询的限制是一样的，都要满足最左前缀的要求（有一种情况例是最左的列被指定为常数，下面是一个简单的示例），其他情况下都需要执行排序操作，而无法利用索引排序。

// 最左列为常数，索引：(date,staff_id,customer_id)

```
select staff_id,customer_id from demo where date = '2015-06-01' order by staff_id,customer_id
```

7、冗余和重复索引

冗余索引是指在相同的列上按照相同的顺序创建的相同类型的索引，应当尽量避免这种索引，发现后立即删除。比如有一个索引(A,B)，再创建索引(A)就是冗余索引。冗余索引经常发生在为表添加新索引时，比如有人新建了索引(A,B)，但这个索引不是扩展已有的索引(A)。

大多数情况下都应该尽量扩展已有的索引而不是创建新索引。但有极少情况下出现性能方面的考虑需要冗余索引，比如扩展已有索引而导致其变得过大，从而其他使用该索引的查询。

8、删除长期未使用的索引

定期删除一些长时间未使用过的索引是一个非常好的习惯。

关于索引这个话题打算就此打住，最后要说一句，索引并不总是最好的工具，只有当索引帮助提高查询速度带来的好处大于其带来的额外工作时，索引才是好的。对于非常小的表，简单的全表扫描更高效。对于中到大型的表，索引就非常有效。对于超大型的表，建立和维护索引的代价随之增长，这时候其他技术也许比分区表。最后的最后，explain后再提测是一种美德。

特定类型查询优化

优化COUNT()查询

COUNT()可能是被大家误解最多的函数了，它有两种不同的作用，其一是统计某个列值的数量，其二是统计行数。统计列值时，要求列值是非空的，它不会统计NULL。如果确认括号中的表达式不可能为空时，实际上就是在统计行数。最简单的就是当使用COUNT(*)时，并不是我们所想象的那样扩展成所有的列，实它会忽略所有的列而直接统计行数。

我们最常见的误解也就在这儿，在括号内指定了一列却希望统计结果是行数，而且还常常误以为前者的性能会更好。但实际并非这样，如果要统计行数，直接用COUNT(*)，意义清晰，且性能更好。

有时候某些业务场景并不需要完全精确的COUNT值，可以用近似值来代替，EXPLAIN出来的行数就是一个不错的近似值，而且执行EXPLAIN并不需要真正执行查询，所以成本非常低。通常来说，执行COUNT(*)都需要扫描大量的行才能获取到精确的数据，因此很难优化，MySQL层面还能做得也就只有覆盖索引了。不还能解决问题，只有从架构层面解决了，比如添加汇总表，或者使用redis这样的外部缓存系统。

优化关联查询

在大数据场景下，表与表之间通过一个冗余字段来关联，要比直接使用JOIN有更好的性能。如果确实需要使用关联查询的情况下，需要特别注意的是：

确保ON和USING字句中的列上有索引。在创建索引的时候就要考虑到关联的顺序。当表A和表B用列c关联的时候，如果优化器关联的顺序是A、B，那2
要在A表的对应列上创建索引。没有用到的索引会带来额外的负担，一般来说，除非有其他理由，只需要在关联顺序中的第二张表的相应列上创建索引
因下文分析）。

确保任何的GROUP BY和ORDER BY中的表达式只涉及到一个表中的列，这样MySQL才有可能使用索引来优化。

要理解优化关联查询的第一个技巧，就需要理解MySQL是如何执行关联查询的。当前MySQL关联执行的策略非常简单，它对任何的关联都执行**嵌套循环关联**
即先在一个表中循环取出单条数据，然后在嵌套循环到下一个表中寻找匹配的行，依次下去，直到找到所有表中匹配的行为为止。然后根据各个表匹配的行，
询中需要的各个列。

太抽象了？以上面的示例来说明，比如有这样的一个查询：

```
SELECT A.xx,B.yy  
FROM A INNER JOIN B USING(c)  
WHERE A.xx IN (5,6)
```

假设MySQL按照查询中的关联顺序A、B来进行关联操作，那么可以用下面的伪代码表示MySQL如何完成这个查询：

```
outer_iterator = SELECT A.xx,A.c FROM A WHERE A.xx IN (5,6);  
outer_row = outer_iterator.next;  
while(outer_row) {  
    inner_iterator = SELECT B.yy FROM B WHERE B.c = outer_row.c;  
    inner_row = inner_iterator.next;  
    while(inner_row) {  
        output[inner_row.yy,outer_row.xx];  
        inner_row = inner_iterator.next;  
    }  
    outer_row = outer_iterator.next;  
}
```



高效运维

TA的个人主页 >

原创

粉丝

获赞

评论

访问

19

284

91

44

24万+

等级: 博客 5

周排名: 6万+

积分: 2866

总排名: 2万+

勋章: 

关注

私信



可视化分析平台

最新文章

灵雀云获英特尔战略投资 加速抢占容器PaaS 制高点

万台服务器跨平台系统补丁自动化部署实践

DevOps 落地，不能只是去学习互联网公司

运维自动化 | 20 条来自学霸的经验总结

决定你上限的，不是能力，而是格局

有时你不得不承认，这个世界上公平的数据是不存在的

归档

2019年5月

10篇

如下的样子：

```
SELECT film.film_id,film.description  
FROM film INNER JOIN (  
    SELECT film_id FROM film ORDER BY title LIMIT 50,5  
    tmp USING(film_id);
```

MySQL扫描尽可能少的页面，获取需要访问的记录后在根据关联列回原表查询所需要的列。

位置，那么下次就可以直接从该书签记录的位置开始扫描，这样就可以避免使用OFFSET，比如下面的查询：

2018年4月28篇

2018年3月37篇

2018年2月24篇

CSDN

首页 博客 学院 下载 论坛 问答 活动 专题 招聘 APP VIP会员 搜博主文章

2017年12月17篇

展开

MySQL处理UNION时策略是元创建临时表，然后再把各个查询结果插入到临时表中，最后再来做查询。因此很多优化策略在UNION查询中都没有Y等字句“下推”到各个子查询中，以便优化器可以充分利用这些条件先优化。

除查以结理理其

热门文章

从零搭建一个自动化运维体系
阅读数 32328

重磅！全世界第一份 AIOps 白皮书（诚意版）独家发布
阅读数 17051

2017年最精彩的自动化运维、智能运维文章都在这了
阅读数 9418

听说你要离开大厂去创业公司做CTO？
阅读数 8993

用 Python 开发一个企业级的监控平台
阅读数 8930

最新评论

腾讯十年运维老兵：运维团队的五个“...
qq_21228639：肖邦主厉害

用 Python 开发一个企业级的...
ailinyingai：很不错的文章

AI 时代下腾讯的海量业务智能监控...
bozy：mark

腾讯十年运维老兵：运维团队的五个“...
u010180165：公司招运维总监，联系18702100525

智能运维就是 由 AI 代替运维人...
weixin_44823919：DCIM就可以实现全生命周期管理,咨询我 xb_xie

理论探索，深度实践，经验分享

2018年最重要的一场 DevOps 技术盛宴

DevOps 国际峰会全面起航

大会亮点

个 DevOps 标准《研发运营一体化能力成熟度模型》全权威解读

领军人物 John Willis 畅聊 DevOps 十年

Ops 国内外互联网、金融、通信、零售等行业的系统性案例

近80个演讲完整覆盖DevOps 全技术领域

op 和Open Space（沉浸式学习和问题研讨）

QQ客服 kefu@csdn.net

客服论坛 400-660-0108

工作时间 8:30-22:00

关于我们 招聘 广告服务 网站地图

京ICP备19004658号 经营性网站备案信息

公安备案号 11010502030143

©1999-2020 北京创新乐知网络技术有限公司

网络110报警服务

北京互联网违法和不良信息举报中心

中国互联网举报中心 家长监护

版权与免责声明 版权申诉



DevOps 国际峰会 暨 DevOps 金融峰会 2018 · 北京

指导单位：OSCAR联盟 主办单位：DevOps时代社区、高效运维社区

2018年6月29-30日 北京悠唐皇冠假日酒店



扫描二维码进入大会官网

点击阅读原文，更多惊喜

点赞 5 收藏 分享 ...



高效运维

发布了19 篇原创文章 · 获赞 91 · 访问量 24万+

私信



毕业论文

写作的朋友-可在线改重的论文查重系统

论文查重知网查重

广告



想对作者说点什么



the_fool_ 1年前 谢谢博主分享~



低调小熊猫 1年前

写这么多，作者辛苦了，我会慢慢看的，不然一下消化不了

我必须得告诉大家的MySQL优化原理

2019独角兽企业重金招聘Python工程师标准>>>

阅读数 28

博文 来自: weixin_340...

【转】我必须得告诉大家的MySQL优化原理

我必须得告诉大家的MySQL优化原理转载于:https://www.cnblogs.com/wangwangfei/p/9172037.html...

阅读数 12

博文 来自: weixin_306...

在中国程序员是青春饭吗？

今年，我也32了，为了不给大家误导，咨询了猎头、圈内好友，以及年过35岁的几位老程序员.....舍了老脸去揭人家伤疤.....

阅读数 16万+


博文 来自: 启舰

Synchronized关键字深析（小白慎入，深入jvm源码，两万字长文）

从jvm层面解析synchronized，看完绝对可以超越绝大多数人

阅读数 7939

博文 来自: Java新生代



订单管理系统

订单管理系统

广告

大学四年，我决定把Java学习过的书籍都分享一遍

给岁月以文明，而不是给文明以岁月，技术人读书我觉得很有必要，那份书单的大部分书我觉得对您有用。...

博文 来自：敖丙

阅读数 1万+

程序员请照顾好自己，周末病魔差点一套带走我。

程序员在一个周末的时间，得了重病，差点当场去世，还好及时挽救回来了。...

博文 来自：敖丙

阅读数 8万+

卸载 x 雷某度！GitHub 标星 1.5w+，从此我只用这款全能高速下载工具！

作者 | Rocky0429来源 | Python空间大家好，我是 Rocky0429，一个喜欢在网上收集各种资源的蒟蒻...网上资源眼花缭乱，...

博文 来自：Rocky0429

阅读数 16万+

为什么猝死的都是程序员，基本上不见产品经理猝死呢？

相信大家时不时听到程序员猝死的消息，但是基本上听不到产品经理猝死的消息，这是为什么呢？我们先百度搜一下：程序...

博文 来自：曹银飞的专栏

阅读数 11万+

毕业5年，我问遍了身边的大佬，总结了他们的学习方法

我问了身边10个大佬，总结了他们的学习方法，原来成功都是有迹可循的。

博文 来自：敖丙

阅读数 15万+

我必须得告诉大家的MySQL优化原理_MayMatrix 的博客-CSDN博客

我必须得告诉大家的 MySQL 优化原理 - 运维派 - CSDN博客



IT运维管理外包

it运维服务外包

广告

推荐10个堪称神器的学习网站

每天都会收到很多读者的私信，问我：“二哥，有什么推荐的学习网站吗？最近很浮躁，手头的一些网站都看烦了，想看看二...

博文 来自：沉默王二

阅读数 25万+

我必须得告诉大家的MySQL优化原理_数据库_心向阳光-CSDN博客

我必须得告诉大家的MySQL优化原理(2)_weixin_33861800..._CSDN博客

阿里程序员写了一个新手都写不出的低级bug，被骂惨了。

这种新手都不会范的错，居然被一个工作好几年的小伙子写出来，差点被当场开除了。...

博文 来自：敖丙

阅读数 16万+



weixin_34077371

4728篇文章

关注 排名:千里之外



MayMatrix

471篇文章

关注 排名:5000+



weixin_30614587

4413篇文章

关注 排名:千里之外



启舰

301篇文章

关注 排名:99

我必须得告诉大家的MySQL的优化原理 - wssjdysf1的专栏 - CSDN博客

我必须得告诉大家的MySQL优化原理 - qq_36210329的博客 - CSDN博客

没用过这些 IDEA 插件？怪不得写代码头疼

使用插件，可以提高开发效率。对于开发人员很有帮助。这篇博客介绍了IDEA中最常用的一些插件。...

博文 来自：扬帆向海的...

阅读数 6万+

AI 没让人类失业，搞 AI 的人先失业了

最近和几个 AI 领域的大佬闲聊根据他们讲的消息和段子改编出下面这个故事如有雷同都是巧合1. 老王创业失败，被限制高消...

博文 来自：CSDN 官...

阅读数 10万+

我必须得告诉大家的MySQL优化原理_S_L_zheng的博客-CSDN博客

我必须得告诉大家的MySQL优化原理2 - weixin_33892359的博客...

作为一名大学生，如何在B站上快乐的学习？

B站是个宝，谁用谁知道????作为一名大学生，你必须掌握的一项能力就是自学能力，很多看起来很牛X的人，你可以了解下...

博文 来自：编码之外的...

阅读数 8万+

https://blog.csdn.net/Mes8Y62b6ogV207/article/details/80238577

页码：13/16



传统ERP已经过时，2019流行的ERP系统是这一款！

国内erp软件排行榜

广告

我必须得告诉大家的MySQL优化原理2 - weixin_34138056的博客...

一个程序在计算机中是如何运行的？超级干货！！

阅读数 10万+

强烈声明：本文很干，请自备茶水！????开门见山，咱不说废话！你有没有想过，你写的程序，是如何在计算机中运行的吗... 博文 来自： 编码之外的...

那个在阿里养猪的工程师，5年了.....

阅读数 3万+

简介： 在阿里，走过1825天，没有趴下，依旧斗志满满，被称为“五年陈”。他们会被授予一枚戒指，过程就叫做“授戒仪式”... 博文 来自： 阿里技术

Java校招入职华为，半年后我跑路了

阅读数 18万+

何来我，一个双非本科弟弟，有幸在 19 届的秋招中得到前东家华为（以下简称 hw）的赏识，当时秋招签订就业协议，说是... 博文 来自： JavaEdge

世界上有哪些代码量很少，但很牛逼很经典的算法或项目案例？

阅读数 3万+

点击上方蓝字设为星标下面开始今天的学习～今天分享四个代码量很少，但很牛逼很经典的算法或项目案例。1、no code 项... 博文 来自： 程序员吴师...

强烈推荐10本程序员必读的书

阅读数 8万+

很遗憾，这个春节注定是刻骨铭心的，新型冠状病毒让每个人的神经都是紧绷的。那些处在武汉的白衣天使们，尤其值得我... 博文 来自： 沉默王二



软文找写手,80一篇

软文发稿网站

广告

非典逼出了淘宝和京东，新冠病毒能够逼出什么？

阅读数 5万+

loonggg读完需要5分钟速读仅需 2 分钟大家好，我是你们的校长。我知道大家在家里都憋坏了，大家可能相对于封闭在家里“... 博文 来自： 非著名程序员

为什么说程序员做外包没前途？

阅读数 9万+

之前做过不到3个月的外包，2020的第一天就被释放了，2019年还剩1天，我从外包公司离职了。我就谈谈我个人的看法吧。... 博文 来自： dotNet全栈...

B 站上有哪些很好的学习资源？

阅读数 13万+

哇说起B站，在小九眼里就是宝藏般的存在，放年假宅在家时一天刷6、7个小时不在话下，更别提今年的跨年晚会，我简直... 博文 来自： 九章算法的...

终于！疫情之下，第一批企业没能熬住面临倒闭，员工被遣散，没能等来春暖花开！

阅读数 4万+

先来看一个图：这个春节，我同所有人一样，不仅密切关注这次新型肺炎，还同时关注行业趋势和企业。在家憋了半个月，... 博文 来自： 码农突围

昂，我24岁了

阅读数 2万+

24岁的程序员，还在未来迷茫，不知道能不能买得起房子 博文 来自： 敖丙

作为程序员的我，大学四年一直自学，全靠这些实用工具和学习网站！

阅读数 7万+

我本人因为高中沉迷于爱情，导致学业荒废，后来高考，毫无疑问进入了一所普普通通的大学，实在惭愧????我又是那么好... 博文 来自： 编码之外的...

新来个技术总监，禁止我们使用Lombok！

阅读数 2万+

我有个学弟，在一家小型互联网公司做Java后端开发，最近他们公司新来了一个技术总监，这位技术总监对技术细节很看重... 博文 来自： HollisChua...

疫情下的招聘季还会是金三银四吗？

阅读数 3万+

想必大家都看过朋友圈流行的一个段子：前天一觉醒来，假期还有⑤天。昨天一觉醒来，假期还有⑦天。今天一觉醒来，假... 博文 来自： 启舰

字节跳动的技术架构

阅读数 1万+

字节跳动创立于2012年3月，到目前仅4年时间。从十几个工程师开始研发，到上百人，再到200余人。产品线由内涵段子，... 博文 来自： 作一个独立...

文档写作利器：Markdown

阅读数 2万+

一、前言无论你是软件开发人员，还是互联网写作者，为了使自己写的文档或作品更好的流通，便于在不同场合、不同环境、... 博文 来自： xcbeyond|...

C++(STL源码)：37---仿函数(函数对象)源码剖析	阅读数 2032
待续	博文 来自： 江南、董少
在三线城市工作爽吗？	阅读数 6万+
我是一名程序员，从正值青春年华的 24 岁回到三线城市洛阳工作，至今已经 6 年有余。一不小心又暴露了自己的实际年龄...	博文 来自： 沉默王二
这些插件太强了，Chrome 必装！尤其程序员！	阅读数 1万+
推荐 10 款我自己珍藏的 Chrome 浏览器插件	博文 来自： 沉默王二
@程序员：GitHub这个项目快薅羊毛	阅读数 1万+
今天下午在朋友圈看到很多人都在发github的羊毛，一时没明白是怎么回事。后来上百度搜索了一下，原来真有这回事，毕...	博文 来自： dotNet全栈...
经典算法（21）毕业生求职必会算法【八皇后问题】	阅读数 1万+
【八皇后问题】就是回溯算法的典型，这篇博客给出了详细的实现逻辑以及完整的代码实现。...	博文 来自： 扬帆向海的...
又一程序员删库跑路了	阅读数 9715
loonggg读完需要2分钟速读仅需 1 分钟今天刷爆朋友圈和微博的一个 IT 新闻，估计有很多朋友应该已经看到了。程序员删...	博文 来自： 非著名程序员
再不跳槽，应届毕业生拿的都比我多了！	阅读数 7682
跳槽几乎是每个人职业生涯的一部分，很多HR说“三年两跳”已经是一个跳槽频繁与否的阈值了，可为什么市面上有很多程序...	博文 来自： 九章算法的...
我以为我学懂了数据结构，直到看了这个导图才发现，我错了	阅读数 7955
数据结构与算法思维导图	博文 来自： 龙跃十二
String s = new String(" a ") 到底产生几个对象？	阅读数 7234
老生常谈的一个梗，到2020了还在争论，你们一天天的，哎哎哎，我不是针对你一个，我是说在座的各位都是人才！上图红...	博文 来自： 宜春
超全Python图像处理讲解（多图预警）	阅读数 2942
文章目录Pillow模块讲解一、Image模块1.1 、打开图片和显示图片1.2、创建一个简单的图像1.3、图像混合（1）透明度混合...	博文
20道你必须会背会的微服务面试题，面试一定会被问到	阅读数 4万+
这篇博客总结了面试中最常见的微服务面试题，相信对你有所帮助。	博文
良心推荐，我珍藏的一些Chrome插件	阅读数 7万+
上次搬家的时候，发了一个朋友圈，附带的照片中不小心暴露了自己的 Chrome 浏览器插件之多，于是就有小伙伴评论说分...	博文
看完这篇HTTP，跟面试官扯皮就没问题了	阅读数 9万+
我是一名程序员，我的主要编程语言是 Java，我更是一名 Web 开发人员，所以我必须要了解 HTTP，所以本篇文章就来带...	博文
HTTP性能极限优化	阅读数 3829
无论你在做前端、后端还是运维，HTTP都是不得不打交道的网络协议。它是最常用的应用层协议，对它的优化，既能通过降...	博文
史上最全的IDEA快捷键总结	阅读数 12万+
现在Idea成了主流开发工具，这篇博客对其使用的快捷键做了总结，希望对大家的开发工作有所帮助。...	博文
谁是华为扫地僧？	阅读数 4万+
是的，华为也有扫地僧！2020年2月11-12日，“养在深闺人不知”的华为2012实验室扫地僧们，将在华为开发者大会2020（Cl...	博文
2020年，冯唐49岁：我给20、30岁IT职场年轻人的建议	阅读数 10万+
点击“技术领导力”关注△每天早上8:30推送 作者 Mr.K 编辑 Emma 来源 技术领导力(ID：jishulingdaoli) 前天的推文《冯唐： ...	博文
最全最强！世界大学计算机专业排名总结！	阅读数 1万+
我正在参与CSDN200进20，希望得到您的支持，扫码续投票5次。感谢您！（为表示感谢，您投票后私信我，我把我总结的...	博文
一份王者荣耀的英雄数据报告	阅读数 8888
咪哥杂谈本篇阅读时间约为 6 分钟。1前言前一阵写了关于王者的一些系列文章，从数据的获取到数据清洗，数据落地，都...	博文

<div><div>推荐一些有趣的在线编程游戏</div><div>1.Robocode 让坦克们互相博弈的游戏，你可以看到它们飞奔，碾碎一切挡道的东西。机器人配有雷达与火炮，选手在躲避...</div></div>	<div>阅读数 1万+</div> <div>博文</div>
<div><div>工作十年的数据分析师被炒，没有方向，你根本躲不过中年危机</div><div>2020年刚刚开始，就意味着离职潮高峰的到来，我身边就有不少人拿着年终奖离职了，而最让我感到意外的，是一位工作十...</div></div>	<div>阅读数 1万+</div> <div>博文</div>
<div><div>那些年，我们信了课本里的那些鬼话</div><div>教材永远都是有错误的，从小学到大学，我们不断的学习了很多错误知识。 斑羚飞渡 在我们学习的很多小学课文里，有很多...</div></div>	<div>阅读数 3万+</div> <div>博文</div>
<div><div>张朝阳回应迟到 1 分钟罚 500：资本家就得剥削员工</div><div>loonggg读完需要2分钟速读仅需 1 分钟大家我，我是你们的校长。前几天，搜狐的董事局主席兼 CEO 张朝阳和搜狐都上热...</div></div>	<div>阅读数 7352</div> <div>博文</div>
<div><div>【蘑菇街技术部年会】程序员与女神共舞，鼻血再次没止住。（文末内推）</div><div>蘑菇街技术部的年会，别开生面，一样全是美女。</div></div>	<div>阅读数 2万+</div> <div>博文</div>
<div><div>为什么程序猿都不愿意去外包？</div><div>分享外包的组织架构，盈利模式，亲身经历，以及根据一些外包朋友的反馈，写了这篇文章，希望对正在找工作的老铁有所...</div></div>	<div>阅读数 3万+</div> <div>博文</div>
<div><div>当你在浏览器中，忘记了曾经的登录密码怎么办...</div><div>咪哥杂谈本篇阅读时间约为 5 分钟。1前言你还在为使用浏览器的时候，忘记密码而烦恼吗？今天要分享的不为人知的小技...</div></div>	<div>阅读数 1390</div> <div>博文</div>
<div><div>两年前不知如何编写代码的我，现在是一名人工智能工程师</div><div>全文共3526字，预计学习时长11分钟 图源：Unsplash 经常有小伙伴私信给小芯，我没有编程基础，不会写代码，如何进入...</div></div>	<div>阅读数 3万+</div> <div>博文</div>
<div><div>Python实战：抓肺炎疫情实时数据，画2019-nCoV疫情地图</div><div>今天，群里白丕老师问如何用python画武汉肺炎疫情地图。白丕老师是研究海洋生态与地球生物的学者，国家重点实验室成...</div></div>	<div>阅读数 8万+</div> <div>博文</div>
<div><div>作为一个程序员，内存的这些硬核知识你必须懂！</div><div>我们之前讲过CPU，也说了CPU和内存的那点儿事儿，今天咱就再来说说有关内存，作为一个程序员，你必须要懂的哪那些硬...</div></div>	<div>阅读数 2万+</div> <div>博文</div>
<div><div>牛逼！一行代码居然能解决这么多曾经困扰我半天的算法题</div><div>春节假期这么长，干啥最好？当然是折腾一些算法题了，下面给大家讲几道一行代码就能解决的算法题，当然，我相信这些...</div></div>	<div>阅读数 2万+</div> <div>博文</div>
<div><div>Spring框架JdbcTemplate介绍</div><div>文章目录一、JdbcTemplate 概述二、创建对象的源码分析三、JdbcTemplate操作数据库 一、JdbcTemplate 概述 在之前的w...</div></div>	<div>阅读数 3万+</div> <div>博文</div>
<div><div>谁说程序员不懂浪漫——我的C语言结婚请柬（附源码）</div><div>前言：但行好事，莫问前程——《增广贤文》 从上学起开始学C++,后面也做过H5，现在做Android。无论是学习用的，还是...</div></div>	<div>阅读数 2万+</div> <div>博文</div>
<div>Java C语言 Python C++ C# Visual Basic .NET JavaScript PHP SQL Go语言 R语言 Assembly language Swift Ruby MATLAB PL/SQL Perl Visual Basic Objective-C Delphi/Object Pascal Unity3D</div>	