48讲B+树：MySQL数据库索引是如何实现的

作为一个软件开发工程师，你对数据库肯定再熟悉不过了。作为主流的数据存储系统，它在我们的业务开发中，有着举足轻重的地位。在工作中，为了加速数据库中数据的查找速度，我们常用的处理思路是，对表中数据创建索引。那你是否思考过，**数据库索引是如何实现的呢？底层使用的是什么数据结构和算法呢？**

算法解析

思考的过程比结论更重要。跟着我学习了这么多节课，很多同学已经意识到这一点，比如Jerry银银同学。我感到很开心。所以，今天的讲解，我会尽量还原这个解决方案的思考过程，让你知其然，并且知其所以然。

1.解决问题的前提是定义清楚问题

如何定义清楚问题呢？除了对问题进行详细的调研，还有一个办法，那就是，通过**对一些模糊的需求进行假设，来限定要解决的问题的范围**。

如果你对数据库的操作非常了解，针对我们现在这个问题，你就能把索引的需求定义得非常清楚。但是，对于大部分软件工程师来说，我们可能只了解一小部分常用的SQL语句，所以，这里我们假设要解决的问题，只包含这样两个常用的需求：

- 根据某个值查找数据，比如select * from user where id=1234；
- 根据区间值来查找某些数据，比如select * from user where id > 1234 and id < 2345。

除了这些功能性需求之外，这种问题往往还会涉及一些非功能性需求，比如安全、性能、用户体验等等。限于专栏要讨论的主要是数据结构和算法，对于非功能性需求，我们着重考虑**性能方面**的需求。性能方面的需求，我们主要考察时间和空间两方面，也就是**执行效率和存储空间**。

在执行效率方面，我们希望通过索引，查询数据的效率尽可能的高；在存储空间方面，我们希望索引不要消耗太多的内存空间。

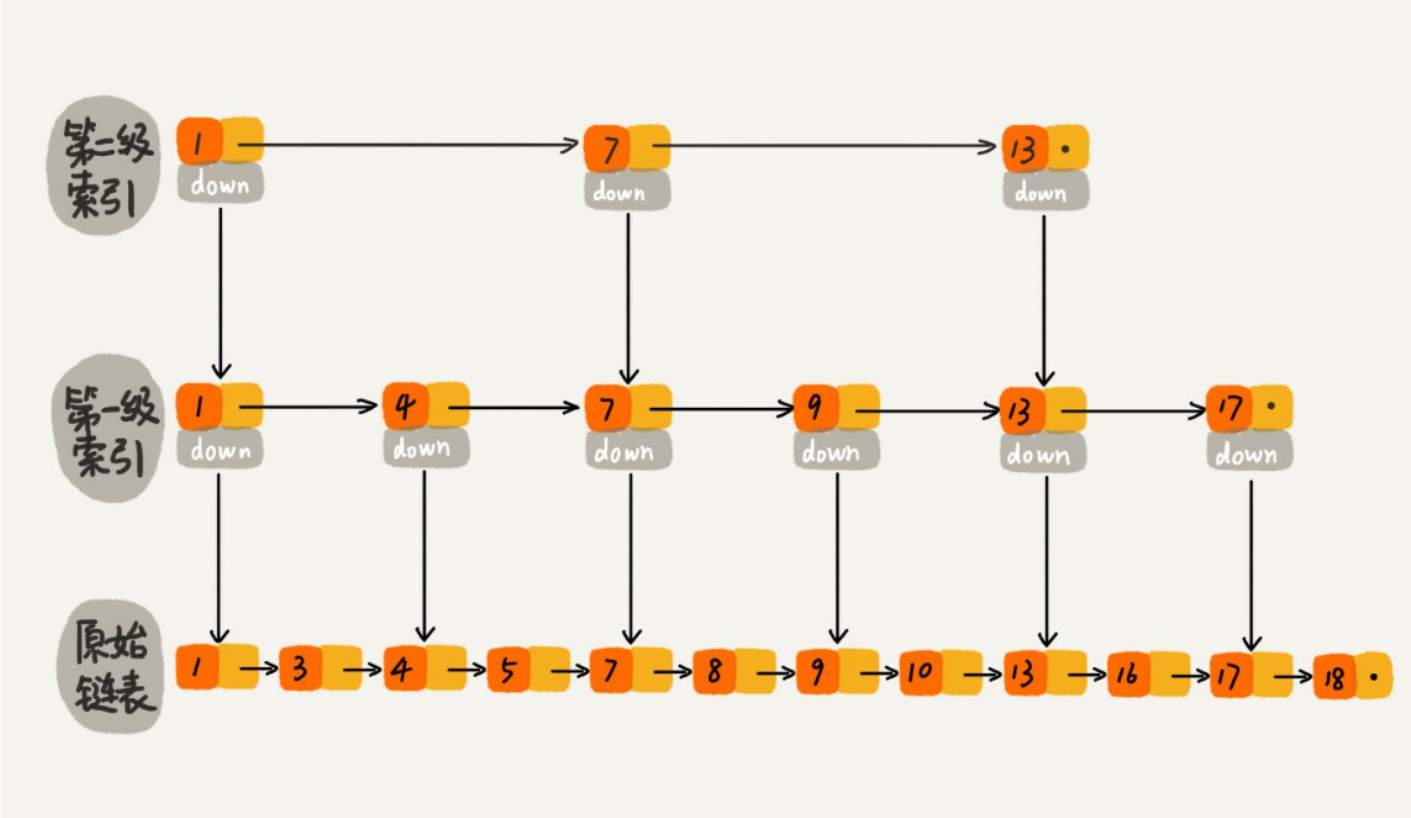
2.尝试用学过的数据结构解决这个问题

问题的需求大致定义清楚了，我们现在回想一下，能否利用已经学习过的数据结构解决这个问题呢？支持快速查询、插入等操作的动态数据结构，我们已经学习过散列表、平衡二叉查找树、跳表。

我们先来看**散列表**。散列表的查询性能很好，时间复杂度是O(1)。但是，散列表不能支持按照区间快速查找数据。所以，散列表不能满足我们的需求。

我们再来看**平衡二叉查找树**。尽管平衡二叉查找树查询的性能也很高，时间复杂度是O(logn)。而且，对树进行中序遍历，我们还可以得到一个从小到大有序的数据序列，但这仍然不足以支持按照区间快速查找数据。

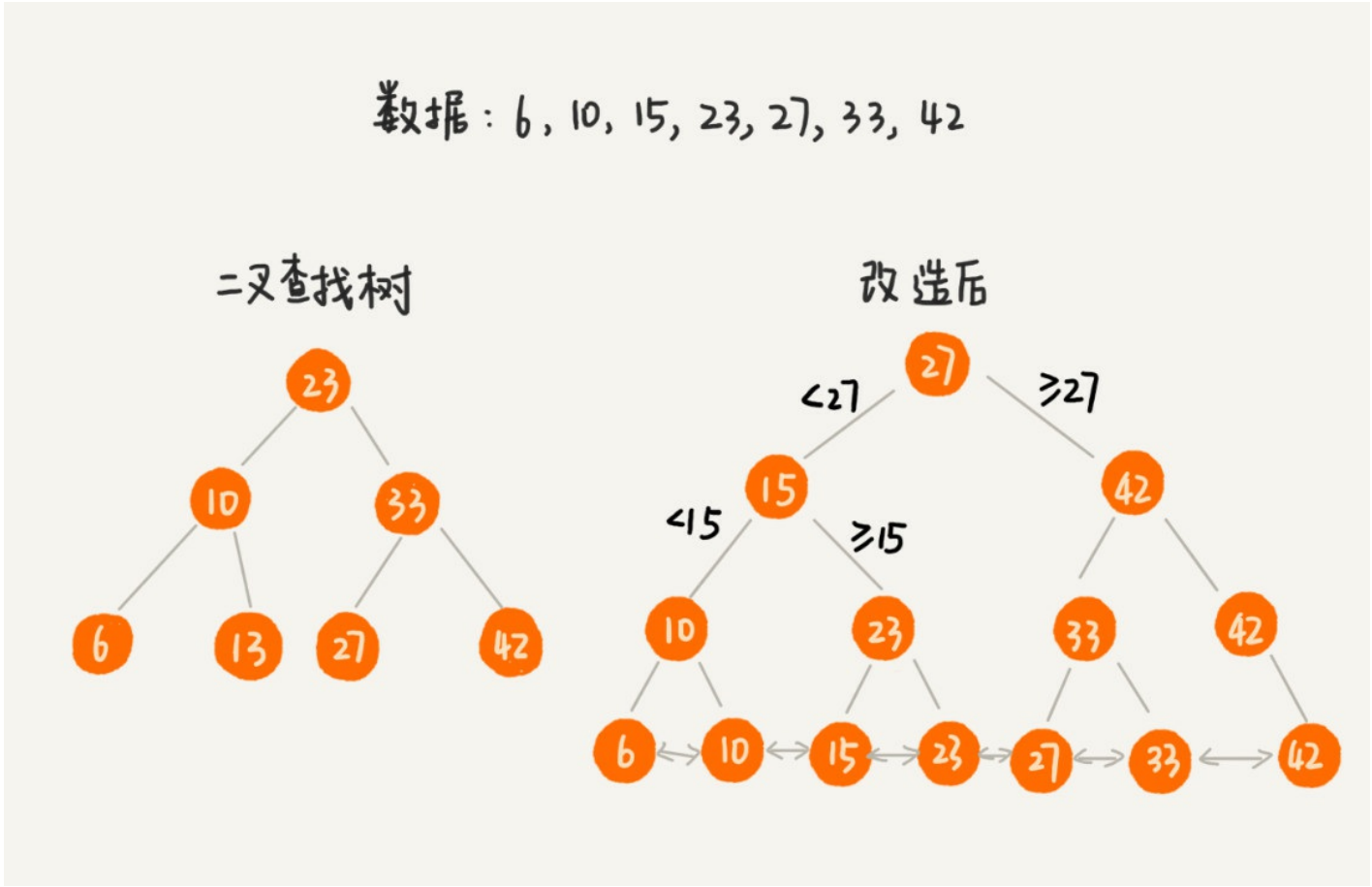
我们再来看**跳表**。跳表是在链表之上加上多层索引构成的。它支持快速地插入、查找、删除数据，对应的时间复杂度是O(logn)。并且，跳表也支持按照区间快速地查找数据。我们只需要定位到区间起点值对应在链表中的结点，然后从这个结点开始，顺序遍历链表，直到区间终点对应的结点为止，这期间遍历得到的数据就是满足区间值的数据。



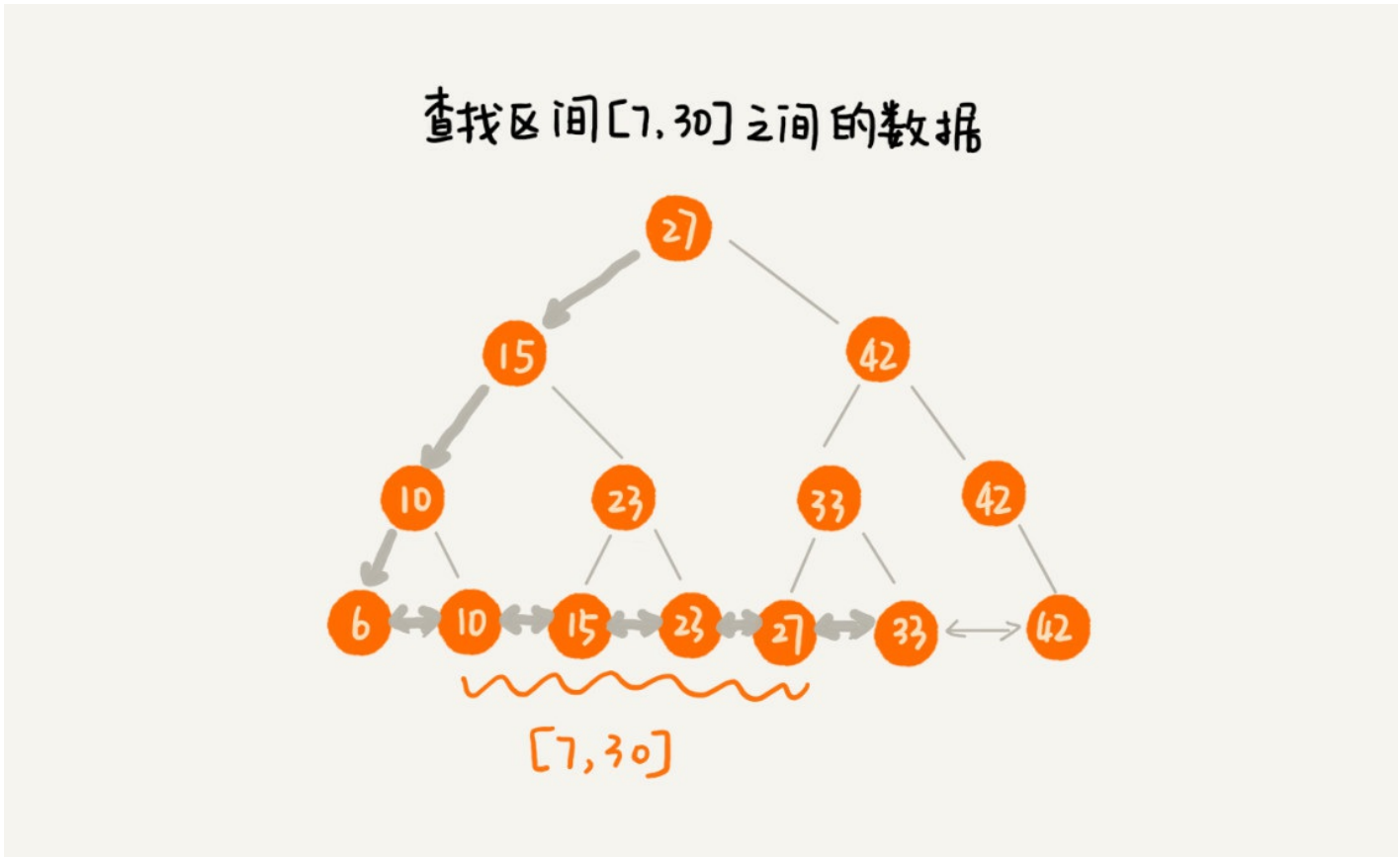
这样看来，跳表是可以解决这个问题。实际上，数据库索引所用到的数据结构跟跳表非常相似，叫作B+树。不过，它是通过二叉查找树演化过来的，而非跳表。为了给你还原发明B+树的整个思考过程，所以，接下来，我还再从二叉查找树讲起，看它是如何一步一步被改造成B+树的。

3.改造二叉查找树来解决这个问题

为了让二叉查找树支持按照区间来查找数据，我们可以对它进行这样的改造：树中的节点并不存储数据本身，而是只是作为索引。除此之外，我们把每个叶子节点串在一条链表上，链表中的数据是从小到大有序的。经过改造之后的二叉树，就像图中这样，看起来是不是很像跳表呢?



改造之后，如果我们要求某个区间的数据。我们只需要拿区间的起始值，在树中进行查找，当查找到某个叶子节点之后，我们再顺着链表往后遍历，直到链表中的结点数据值大于区间的终止值为止。所有遍历到的数据，就是符合区间值的所有数据。



但是，我们要为几千万、上亿的数据构建索引，如果将索引存储在内存中，尽管内存访问的速度非常快，查询的效率非常高，但是，占用的内存会非常多。

比如，我们给一亿个数据构建二叉查找树索引，那索引中会包含大约1亿个节点，每个节点假设占用16个字节，那就需要大约1GB的内存空间。给一张表建立索引，我们需要1GB的内存空间。如果我们要给10张表建立索引，那对内存的需求是无法满足的。如何解决这个索引占用太多内存的问题呢?

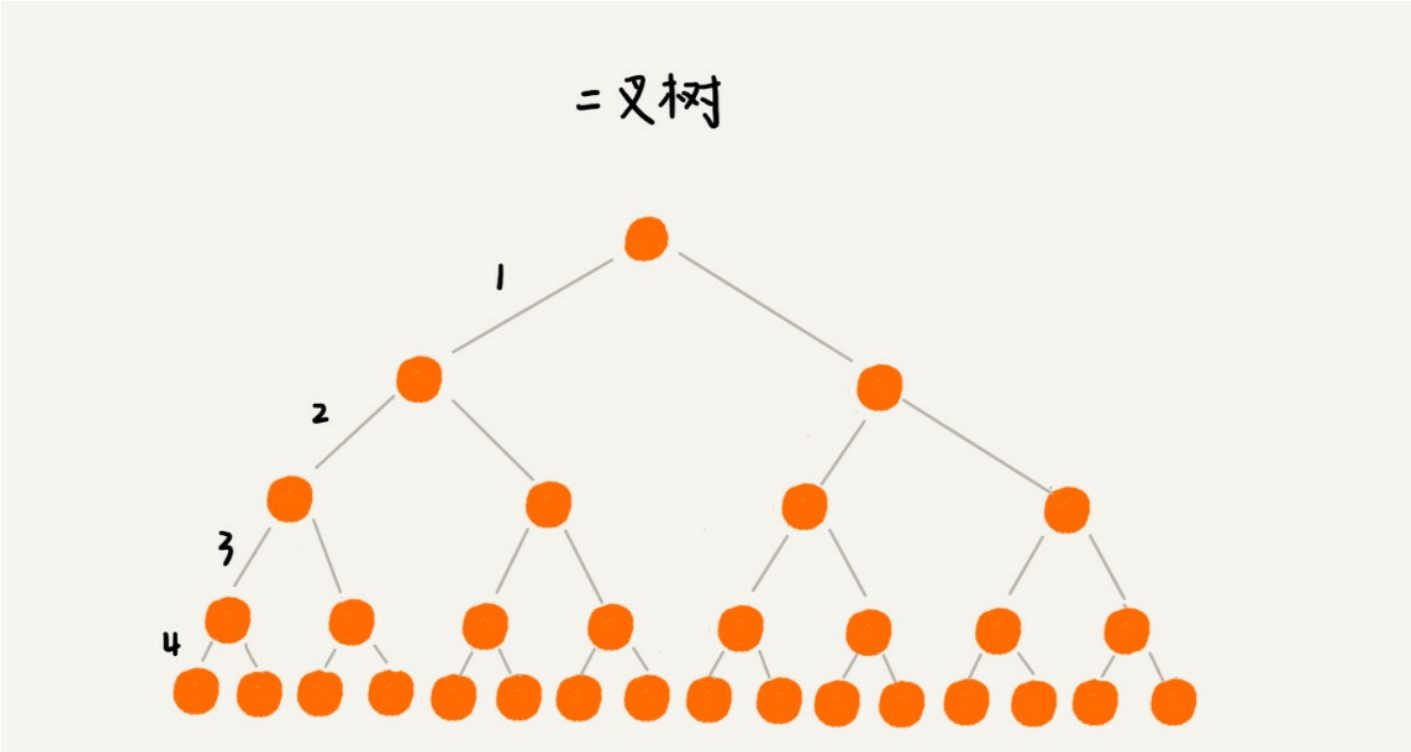
我们可以借助时间换空间的思路，把索引存储在硬盘中，而非内存中。我们都知道，硬盘是一个非常慢速的存储设备。通常内存的访问速度是纳秒级别的，而磁盘访问的速度是毫秒级别的。读取同样大小的数据，从磁盘中读取花费的时间，是从内存中读取所花费时间的上万倍，甚至几十万倍。

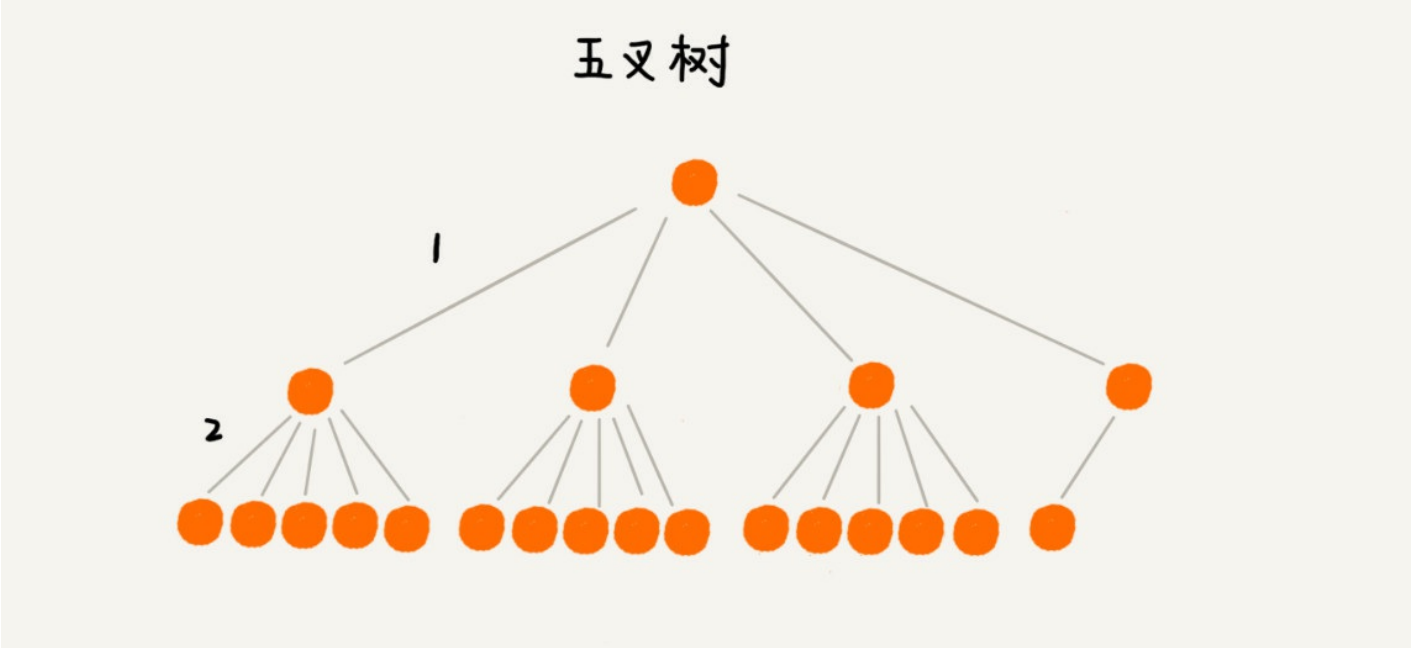
这种将索引存储在硬盘中的方案，尽管减少了内存消耗，但是在数据查找的过程中，需要读取磁盘中的索引，因此数据查询效率就相应降低很多。

二叉查找树，经过改造之后，支持区间查找的功能就实现了。不过，为了节省内存，如果把树存储在硬盘中，那么每个节点的读取（或者访问），都对应一次磁盘IO操作。树的高度就等于每次查询数据时磁盘IO操作的次数。

我们前面讲到，比起内存读写操作，磁盘IO操作非常耗时，所以我们优化的重点就是尽量减少磁盘IO操作，也就是，尽量降低树的高度。那如何降低树的高度呢?

我们来看下，如果我们把索引构建成m叉树，高度是不是比二叉树要小呢？如图所示，给16个数据构建二叉树索引，树的高度是4，查找一个数据，就需要4个磁盘IO操作（如果根节点存储在内存中，其他结点存储在磁盘中），如果对16个数据构建五叉树索引，那高度只有2，查找一个数据，对应只需要2次磁盘操作。如果m叉树中的m是100，那对一亿个数据构建索引，树的高度也只是3，最多只要3次磁盘IO就能获取到数据。磁盘IO变少了，查找数据的效率也就提高了。





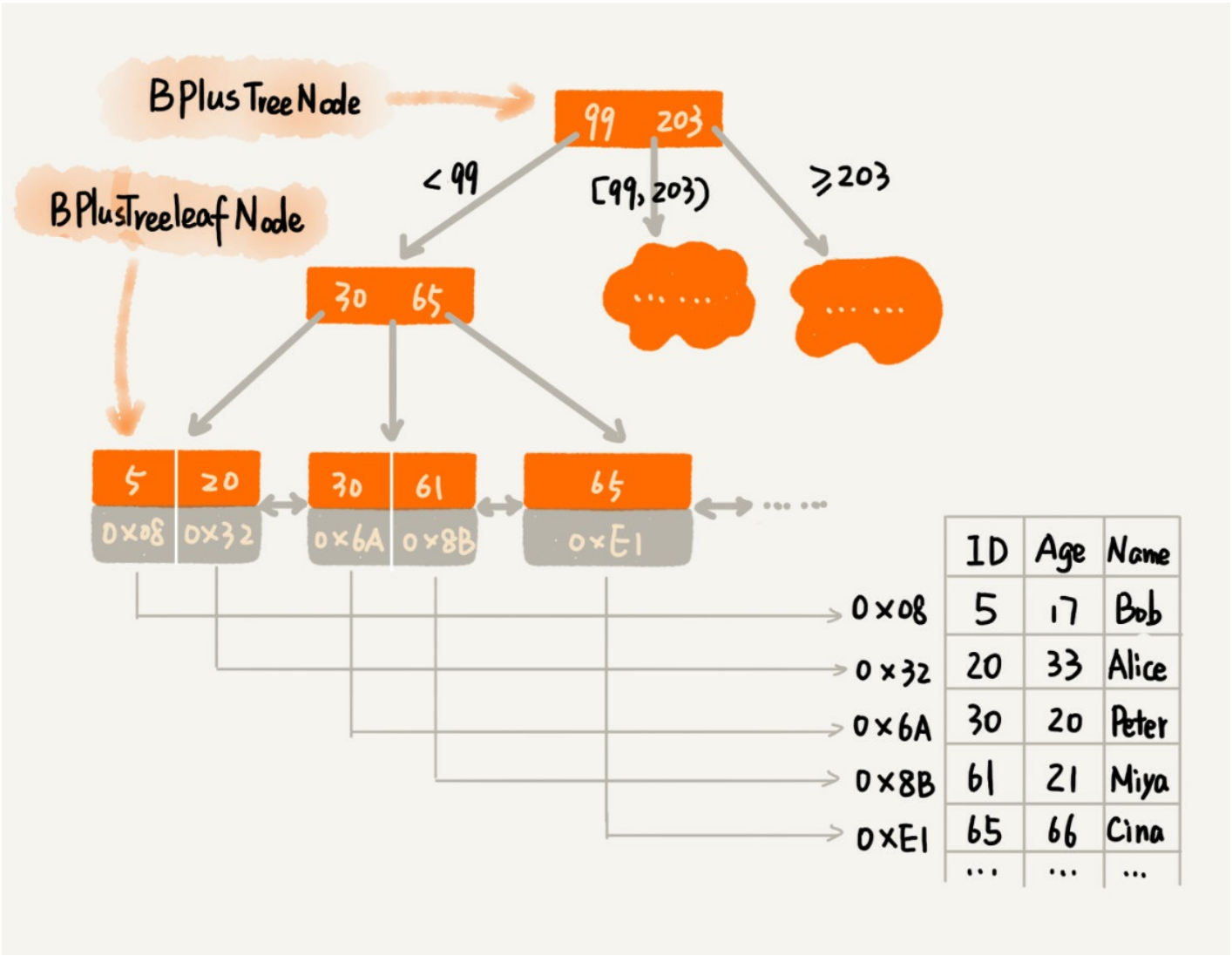
如果我们将m叉树实现B+树索引，用代码实现出来，就是下面这个样子（假设我们给int类型的数据库字段添加索引，所以代码中的keywords是int类型的）：

```
/**
 * 这是B+树非叶子节点的定义。
 *
 * 假设keywords=[3, 5, 8, 10]
 * 4个键值将数据分为5个区间：(-INF,3), [3,5), [5,8), [8,10), [10,INF)
 * 5个区间分别对应：children[0]...children[4]
 *
 * m值是事先计算得到的，计算的依据是让所有信息的大小正好等于页的大小：
 * PAGE_SIZE = (m-1)*4[keywordss大小]+m*8[children大小]
 */
public class BPlusTreeNode {
  public static int m = 5; // 5叉树
  public int[] keywords = new int[m-1]; // 键值，用来划分数据区间
  public BPlusTreeNode[] children = new BPlusTreeNode[m];//保存子节点指针
}

/**
 * 这是B+树中叶子节点的定义。
 *
 * B+树中的叶子节点跟内部结点是不一样的，
 * 叶子节点存储的是值，而非区间。
 * 这个定义里，每个叶子节点存储3个数据行的键值及地址信息。
 *
 * k值是事先计算得到的，计算的依据是让所有信息的大小正好等于页的大小：
 * PAGE_SIZE = k*4[keyw..大小]+k*8[dataAd..大小]+8[prev大小]+8[next大小]
 */
public class BPlusTreeLeafNode {
  public static int k = 3;
  public int[] keywords = new int[k]; // 数据的键值
  public long[] dataAddress = new long[k]; // 数据地址

  public BPlusTreeLeafNode prev; // 这个结点在链表中的前驱结点
  public BPlusTreeLeafNode next; // 这个结点在链表中的后继结点
}
```

我稍微解释一下这段代码。

对于相同个数的数据构建m叉树索引，m叉树中的m越大，那树的高度就越小，那m叉树中的m是不是越大越好呢？到底多大才最合适呢?

不管是内存中的数据，还是磁盘中的数据，操作系统都是按页（一页大小通常是4KB，这个值可以通过getconfig PAGE_SIZE命令查看）来读取的，一次会读一页的数据。如果要读取的数据量超过一页的大小，就会触发多次IO操作。所以，我们在选择m大小的时候，要尽量让每个节点的大小等于一个页的大小。读取一个节点，只需要一次磁盘IO操作。

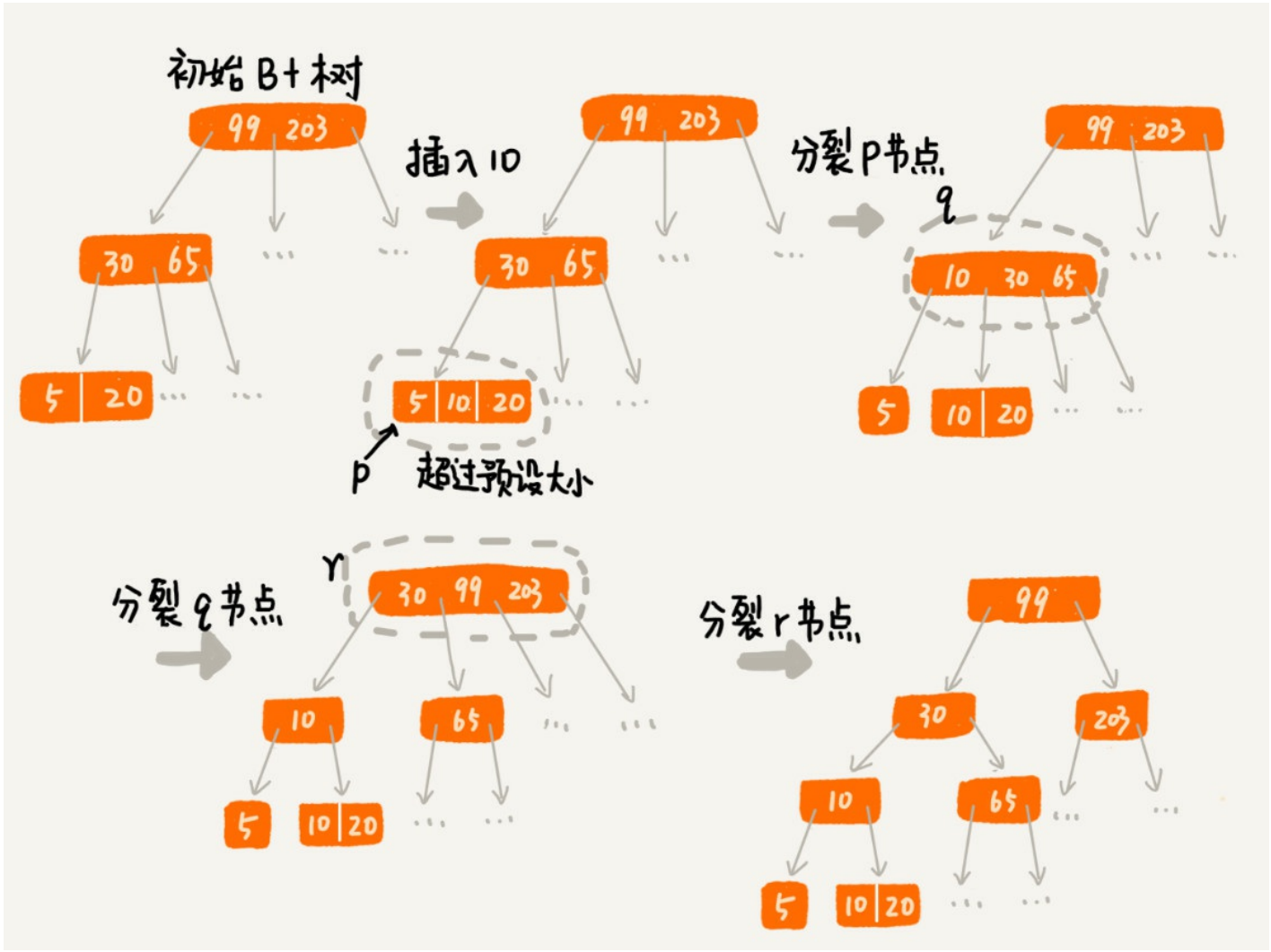


尽管索引可以提高数据库的查询效率，但是，作为一名开发工程师，你应该也知道，索引有利也有弊，它也会让写入数据的效率下降。这是为什么呢?

数据的写入过程，会涉及索引的更新，这是索引导致写入变慢的主要原因。

对于一个B+树来说，m值是根据页的大小事先计算好的，也就是说，每个节点最多只能有m个子节点。在往数据库中写入数据的过程中，这样就有可能使索引中某些节点的子节点个数超过m，这个节点的大小超过了一个页的大小，读取这样一个节点，就会导致多次磁盘IO操作。我们该如何解决这个问题呢?

实际上，处理思路并不复杂。我们只需要将这个节点分裂成两个节点。但是，节点分裂之后，其上层父节点的子节点个数就有可能超过m个。不过这也没关系，我们可以用同样的方法，将父节点也分裂成两个节点。这种级联反应会从下往上，一直影响到根节点。这个分裂过程，你可以结合着下面这个图一块看，会更容易理解（图中的B+树是一个三叉树。我们限定叶子节点中，数据的个数超过2个就分裂节点；非叶子节点中，子节点的个数超过3个就分裂节点）。

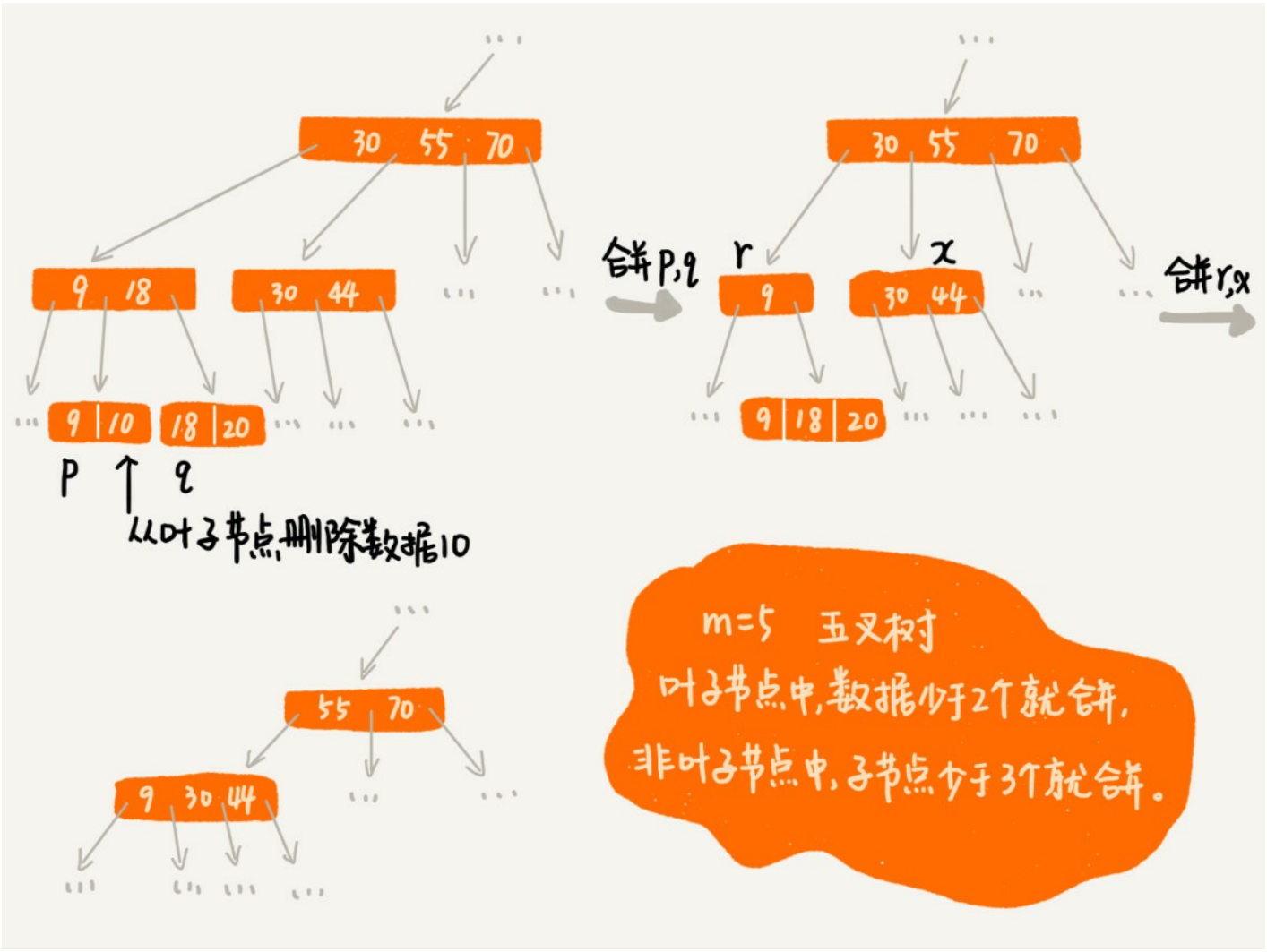


正是因为要时刻保证B+树索引是一个m叉树，所以，索引的存在会导致数据库写入的速度降低。实际上，不光写入数据会变慢，删除数据也会变慢。这是为什么呢?

我们在删除某个数据的时候，也要对应的更新索引节点。这个处理思路有点类似跳表中删除数据的处理思路。频繁的数据删除，就会导致某些结点中，子节点的个数变得非常少，长此以往，如果每个节点的子节点都比较少，势必会影响索引的效率。

我们可以设置一个阈值。在B+树中，这个阈值等于m/2。如果某个节点的子节点个数小于m/2，我们就将它跟相邻的兄弟节点合并。不过，合并之后结点的子节点个数有可能会超过m。针对这种情况，我们可以借助插入数据时候的处理方法，再分裂节点。

文字描述不是很直观，我举了一个删除操作的例子，你可以对比着看下（图中的B+树是一个五叉树。我们限定叶子节点中，数据的个数少于2个就合并节点；非叶子节点中，子节点的个数少于3个就合并节点。）。



数据库索引以及B+树的由来，到此就讲完了。你有没有发现，B+树的结构和操作，跟跳表非常类似。理论上讲，对跳表稍加改造，也可以替代B+树，作为数据库的索引实现的。

B+树发明于1972年，跳表发明于1989年，我们可以大胆猜想下，跳表的作者有可能就是受了B+树的启发，才发明出跳表来的。不过，这个也无从考证了。

总结引申

今天，我们讲解了数据库索引实现，依赖的底层数据结构，B+树。它通过存储在磁盘的多叉树结构，做到了时间、空间的平衡，既保证了执行效率，又节省了内存。

前面的讲解中，为了一步一步详细地给你介绍B+树的由来，内容看起来比较零散。为了方便你掌握和记忆，我这里再总结一下B+树的特点：

- 每个节点中子节点的个数不能超过m，也不能小于m/2；
- 根节点的子节点个数可以不超过m/2，这是一个例外；
- m叉树只存储索引，并不真正存储数据，这个有点儿类似跳表；
- 通过链表将叶子节点串联在一起，这样可以方便按区间查找；
- 一般情况，根节点会被存储在内存中，其他节点存储在磁盘中。

除了B+树，你可能还听说过B树、B-树，我这里简单提一下。实际上，B-树就是B树，英文翻译都是B-Tree，这里的“-”并不是相对B+树中的“+”，而只是一个连接符。这个很容易误解，所以我强调下。

而B树实际上是低级版的B+树，或者说B+树是B树的改进版。B树跟B+树的不同点主要集中在这几个地方：

- B+树中的节点不存储数据，只是索引，而B树中的节点存储数据；
- B树中的叶子节点并不需要链表来串联。

也就是说，B树只是一个每个节点的子节点个数不能小于m/2的m叉树。

课后思考

1. B+树中，将叶子节点串起来的链表，是单链表还是双向链表？为什么？
2. 我们对平衡二叉查找树进行改造，将叶子节点串在链表中，就支持了按照区间来查找数据。我们在散列表（下）讲到，散列表也经常跟链表一块使用，如果我们把散列表中的结点，也用链表串起来，能否支持按照区间查找数据呢？

欢迎留言和我分享，也欢迎点击“请朋友读”，把今天的内容分享给你的好友，和他一起讨论、学习。

精选留言

Jerry银银

听专栏，听到了自己的名字，不敢相信，看了文稿，确实是自己。真是受宠若惊！

2019-01-16 14:32

城

1.链表是双向链表，用以支持前后遍历

2.散列表的节点用链表串起来，并不能实现范围查询，因为散列表本身无序，而B+树是基于二叉查找树演变而成，是有序的

2019-01-16 08:46

Jerry银银

个人觉得B+tree理解起来真不难，抓住几个要点就可以了：

1. 理解二叉查找树
2. 理解二叉查找树会出现不平衡的问题（红黑树理解了，对于平衡性这个关键点就理解了）
3. 磁盘IO访问太耗时
4. 当然，链表知识跑不了 —— 别小瞧这个简单的数据结构，它是链式结构之母
5. 最后，要知道典型的应用场景：数据库的索引结构的设计

还记得，在学生时代，不好好学数据结构的我，当看到这个高大尚的名词“B+tree”时，我心里无比惊慌：这东西貌似不简单。^_^ 那时，也有着王争老师说的这种情况：B-tree，这是B减树；肯定还有个正常的B树；B+tree，这是B加树；然后在我的脑海里面，想当然地认为，它们之间有着这样的大小关系：B-tree < B tree < B+tree

---

对于思考题，@老杨 大哥的回答我觉得很到位了。 我只做一下补充：
第一题： 对于B+tree叶子节点，是用双向链表还是用单链表，得从具体的场景思考。我想，大部分同学在开发中遇到的数据库查询，都遇到过升序或降序问题，即类似这样的sql: select name,age, ... from where uid > startValue and uid < endValue order by uid asc(或者desc)，此时，数据底层实现有两种做法：
1）保证查出来的数据就是用户想要的顺序
2）不保证查出来的数据的有序性，查出来之后再排序
以上两种方案，不加思考，肯定选第一种，因为第二种做法浪费了时间（如果选用内存排序，还是考虑数据的量级）。那如何能保证查询出来的数据就是有序的呢？单链表肯定做不到，只能从头往后遍历，再想想，只能选择双向链表了。此时，可能有的同学又问了：双向链表，多出来了一倍的指针，不是会多占用空间嘛？ 答案是肯定的。可是，我们再细想下，数据库索引本身都已经在磁盘中了，对于磁盘来说，这点空间已经微不足道了，用这点空间换来时间肯定划算呀。顺便提一下：在实际工程应用中，双向链表应用的场景非常广泛，毕竟能大量减少链表的遍历时间

第二题：
答案是「肯定的」。如同@老杨 大哥说的，JDK中的LinkedHashMap为了能做到保持节点的顺序（插入顺序或者访问顺序），就是用双向链表将节点串起来的。 其实，王争老师在《散列表(下）》那一堂课中就已经深入讲解了LinkedHashMap，如果理解了那篇，这个问题应该不难。

最后，我发现王争老师布置的这些课后思考题，都涉及到了之前学到的内容，不知道是有意还是无意的，嘻嘻！

这节的思考题花了蛮多时间进行思考，才能给出以上答案，希望王争老师帮看看是否有不对的地方，谢谢！

2019-01-19 20:59

茴香根

好开心，终于搞清楚经常见到的b+树结构了。从这一节看到对于大数据情况下，m的大小对查询速度有重要影响。如在一些一些特定场合是否可以通过增大内存页和磁盘页大小来进一步提升查询效率。对于思考题中hash做索引，我认为是可行的，但每次更新索引时，如果新进入的节点索引需要插入到相应的位置，要保持叶子链表的有序。

2019-01-16 08:30

老杨同志

问题一，双向链表，方便asc和desc。
问题二，可以支持区间查询。java中linkedHashMap就是链表链表+HashMap的组合，用于实现缓存的lru算法比较方便，不过要支持区间查询需要在插入时维持链表的有序性，复杂度O(n).效率比跳表和b+tree差

2019-01-17 08:25

有朋自远方来

1.利用磁盘预读功能2.主簇索引
觉得这两点也很重要。

2019-01-16 19:42

feifei

老师，看了你的讲解，对于B+树的原理，我基本理解了，我又找了b+树的代码实现，也搞懂怎么回事了，当我看懂了，这个B+树的实现了之后，我就有个问题，这个B+树该如何保存到磁盘中呢？我搜索了好多，也没有找到相关的一个代码，你有这相

关的资料吗？这种数据一般是如何保存的？谢谢

2019-01-24 08:58

作者回复

我懂你的意思。具体我没研究过。我觉得可以直接存到文件里。节点在文件里的位置表示指针。我瞎猜的：）等我研究研究再说：）

2019-01-25 16:30

Monday

请问：

第一段代码，第9行：

PAGE_SIZE = (m-1)*4[keywordss 大小]+m*8[children 大小]

1，这个8指的是引用（指针）占的内存大小吗？

2，引用大小是怎么计算的？和机器是多少位的有什么关系吗？

望争哥回复，谢谢！

2019-01-20 08:23

Monday

先回答思考题：

1. 双向链表，为了支持在O(logn)时间复杂度删除节点

2.支持按区间查找数据。那么问题来了，为什么mysql索引不采用散列表+双向链表的数据结果来实现呢？

2019-01-17 22:09

刘章周

我觉得是双向链表，sql语句中有按照从大到小进行排序，当使用索引进行排序时，如果是单向链表，还要把数据取出来放入内存中排序，效率降低，如果排序的数据较多，内存不够，还会借助外部文件通过归并排序进行排序，效率很低。不知道说的对不对。

2019-01-16 09:11

五岳寻仙

课后思考题1

我觉得应该是双链表。对于区间查找，我们既需要支持大于某个值的查找(向右遍历)，也需要支持小于某个值的查找(向左遍历)。

思考题2

我觉得不可以。因为散列表结点存储的数据是无序的。

2019-01-16 08:33

yaya

1从图上来看，b+叶结点串起来的是双向链表

2不可以，因为散列表的是被mod后的，查询区间依然需要遍历所有结点

以前学b+树的时候，完全不知道它为什么这样设计，感觉很奇怪，今天才明白是为了提供区间查询，优化操作次数。

2019-01-16 08:29

K战神

打卡

2019-01-16 08:19

田伟 คิดถึง

B+树和跳表很像，都是双向链表+索引的结构，数据都放在最下边，利用二分查找进行有序数列查找，区别是啥？我猜想主要区别在索引：

1.高度：同数量级的数据，跳表索引的高度会很高，IO读取次数多，影响查询性能

2.页空间浪费：mysql默认页空间16K,跳表默认一个节点只存一个数，其他空间都浪费了

2019-01-29 18:49

睡痴儿

那块b+树添加的部分如果是二叉树，且限定条件是节点的子节点都是两个的话，会出现最顶层有两个节点的情况啊

2019-01-26 21:44

睡痴儿

双向链表，这样方便查询。

也可以

2019-01-26 11:04

呆梨

打卡，老师从二叉查找树讲到了b树的由来，感觉以后再也不会忘记b树了，讲的很好

2019-01-23 13:53

伟忠

1，双向链表，需要支持大于，小于各种区间查询模式

2，可以，但维护索引，删除数据时间复杂度变高了，哈希只能精确定位，不支持区间查找。

2019-01-22 09:18

每天晒白牙

思考题1:双向链表，为了支持增序和降序

思考题2:可以支持范围查询，但需要保证跳表的数据有序

mysql数据库的索引一直是重点，这篇文章值得多读

2019-01-21 08:34

zixuan

可以思考下B+树为什么不用考虑平衡的问题

2019-01-19 13:29