重生之 MySQL B+Tree 提前问世二十年，MySQL之父叫我师父

原创 李健青@码哥字节 码哥跳动 2025年04月07日 08:29 广东

码哥跳动
《Redis 高手心法》作者，后端架构师，精通Java与Go，宗旨是拥抱技术和对象，面向人...
239篇原创内容

公众号

重生 MySQL 系列第六篇《我重生后，B+树提前问世二十年》。

○ 《重生 MySQL》历史回顾 ○

重生之MySQL 索引失效六大陷阱

重生之我用 2025 年的 InnoDB 知识在 2003 年 IT 圈打工

MySQL MyISAM引擎是什么？有什么致命缺陷？为何现在都不使用了？

MySQL：MyISAM锁表致千万损失！穿越工程师如何逆天改命

MySQL是什么？它的架构是怎样的？假如让你重新设计，你要怎么做？

林渊盯着监控大屏上的红色警报：`[ERROR] Product category tree query timeout after 30s` 商品分类树查询超时！

这已经是本周第三次故障。他打开调试日志，发现每次查询需要遍历 17 层二叉树。

机房的温度飙升到 42 度，磁盘阵列的 LED 指示灯疯狂闪烁。

> 林渊心想：是时候展示我的技术了，这个时代 B+ tree 还未产生。

另一边有人问道：“二叉树 O(logN)的时间复杂度是教科书写的啊！怎么这样？”

进入正文前，介绍下我的点击查看详细介绍 -> **《Java 面试高手心法 58 讲》**专栏内容涵盖 Java 基础、Java 高级进阶、Redis、MySQL、消息中间件、微服务架构设计等面试必考点、面试高频点。

Java 面试高手心法 58 讲

码哥跳动

191 订阅 | 29 内容

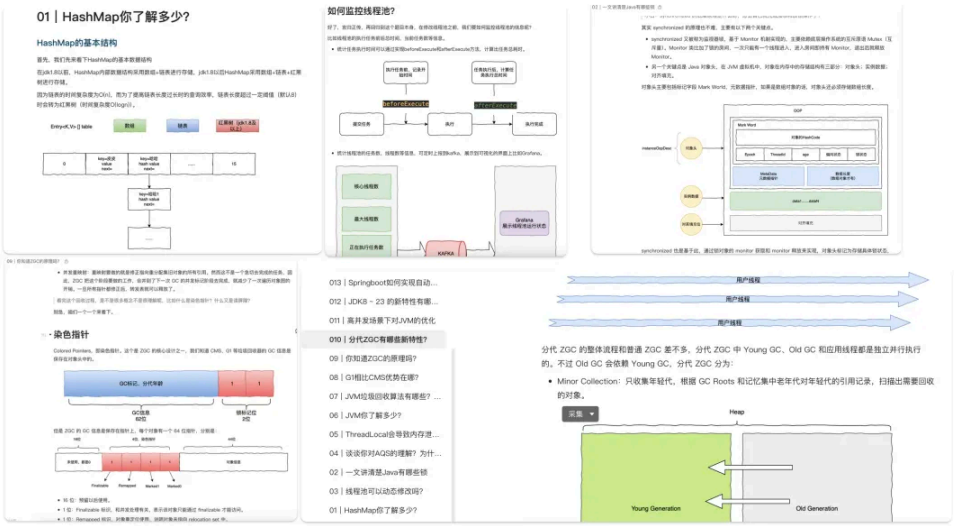
作者：著有《Redis 高手心法》，专栏结合大厂高并发项目的场景，提取作者多年的工作和资深面试官经验。每篇文章平均画了 5 张图，让你高效的学习面试技术要点，掌握互联网 Java 流行技术体系要点难点，系统化的提升技术，做到事半功倍，拿下高薪 Offer。

丢掉你收藏的「面试宝典」，因为它们大多数深度不够，甚至内容还有错误，这也是为何每次面试你都回答不好的原因。原价 128，早鸟价19

微信扫码查看完整内容

现价: ￥ 19

长按扫码购买



正文开始.....

二叉树的致命缺陷

场景还原：商品分类表`prod_category`的父 ID 字段索引。

```
CREATE TABLE prod_category (
  id INT PRIMARY KEY,
  parent_id INT,
  INDEX idx_parent (parent_id) -- 二叉树实现
);
```

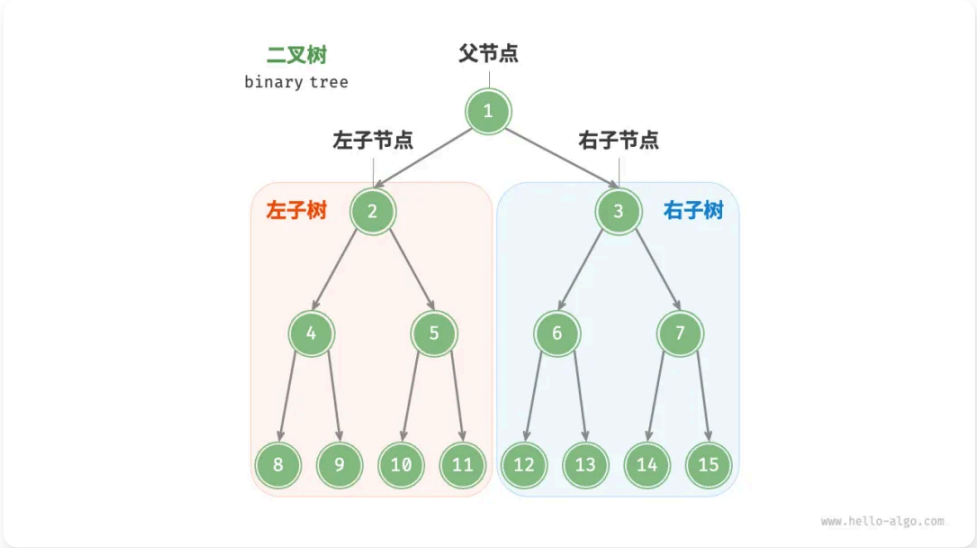
性能灾难：当层级超过 10 层时，查询子类目需要递归遍历：

```
// 原始递归代码（林渊的注释版）
public List<Long> findChildren(Long parentId) {
  // WARNING! 每次递归都是一次磁盘寻道（8ms）
  List<Long> children = jdbc.query("SELECT id FROM prod_category WHERE parent_id=?", p
  for (Long child : children) {
    children.addAll(findChildren(child)); // 指数级IO爆炸
  }
  return children;
}
```

林渊的实验数据（2004 年实验报告节选）：

数据量	树高	平均查询时间
1 万	14	112ms
10 万	17	136ms
100 万	20	160ms

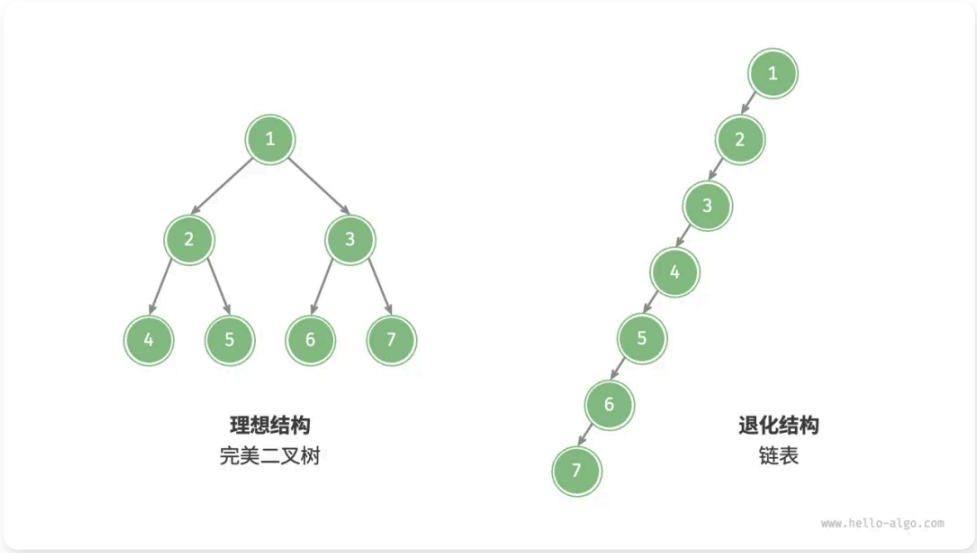
二叉树的机械困境与复杂度陷阱



1970 年代，二叉搜索树（BST）的理论时间复杂度 O(logN)掩盖了物理实现的致命缺陷。以机械硬盘为例：

- **树高灾难**：100 万数据产生约 20 层高度（$\log_2(1{,}000{,}000)=19.9$），假设每次 IO 耗时 8ms，单次查询需 160ms
- **局部性原理失效**：随机磁盘寻道导致缓存命中率趋近于 0。

另外，当所有节点都偏向一侧时，二叉树退化为“链表”。



B Tree

> 林渊的竞争对手 Jake 说：我设计了一个 B Tree，相信是绝世无双的设计。

B 树通过多路平衡设计解决了磁盘 IO 问题。根据《数据库系统实现》的公式推导，B 树的阶数 m 与磁盘页大小的关系为：

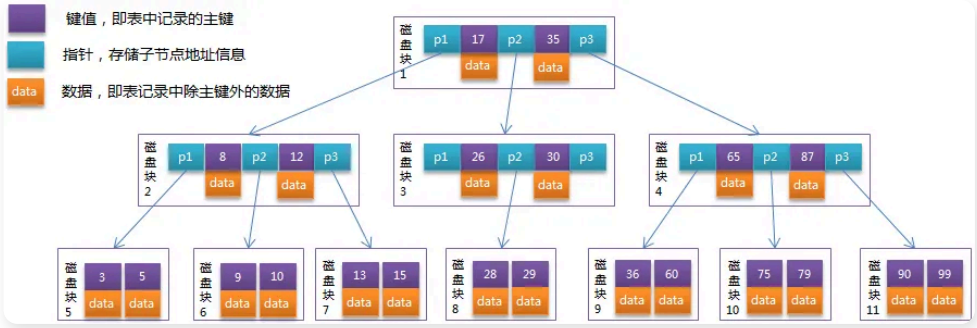
```
m ≥ (PageSize - PageHeader) / (KeySize + PointerSize)
```

以 MySQL 默认 16KB 页为例（PageHeader 约 120 字节，KeySize 8 字节，Pointer 6 字节），阶数 $m \approx (16384-120)/(8+6)=1162$。

10 万数据量下，B 树高度仅需 2 层（$\log_{1162}(100000)\approx 2$），查询 IO 次数从 17 次降为 3 次（根节点常驻内存），总延迟 24ms。

首先定义一条记录为一个二元组[key, data]，key 为记录的键值，对应表中的主键值，data 为一行记录中除主键外的数据。对于不同的记录，key 值互不相同。

B-Tree 中的每个节点根据实际情况可以包含大量的关键字信息和分支，如下图所示为一个 3 阶的 B-Tree：



每个节点占用一个盘块的磁盘空间，一个节点上有两个升序排序的关键字和三个指向子树根节点的指针，指针存储的是子节点所在磁盘块的地址。

两个关键词划分成的三个范围域对应三个指针指向的子树的数据的范围域。

以根节点为例，关键字为 17 和 35，P1 指针指向的子树的数据范围为小于 17，P2 指针指向的子树的数据范围为 17~35，P3 指针指向的子树的数据范围为大于 35。

模拟查找关键字 29 的过程：

1. 根据根节点找到磁盘块 1，读入内存。【磁盘 I/O 操作第 1 次】
2. 比较关键字 29 在区间（17,35），找到磁盘块 1 的指针 P2。
3. 根据 P2 指针找到磁盘块 3，读入内存。【磁盘 I/O 操作第 2 次】
4. 比较关键字 29 在区间（26,30），找到磁盘块 3 的指针 P2。
5. 根据 P2 指针找到磁盘块 8，读入内存。【磁盘 I/O 操作第 3 次】
6. 在磁盘块 8 中的关键字列表中找到关键字 29。

分析上面过程，发现需要 3 次磁盘 I/O 操作，和 3 次内存查找操作。由于内存中的关键字是一个有序表结构，可以利用二分法查找提高效率。

B+Tree

> 林渊反驳：“每个节点中不仅包含数据的 key 值，还有 data 值。
>
> 而每一个页的存储空间是有限的，如果 data 数据较大时将会导致每个节点（即一个页）能存储的 key 的数量很小，当存储的数据量很大时同样会导致 B-Tree 的深度较大，增大查询时的磁盘 I/O 次数，进而影响查询效率。”

而且 BTree 的非叶子节点存储数据，导致范围查询需要跨层跳跃。

林渊脑海中立马翻阅在 2025 年学到的 B+Tree 数据结构，在《MySQL 内核：InnoDB 存储引擎》中发现这段代码：

```
// storage/innobase/btr/btr0btr.cc
void btr_cur_search_to_nth_level(...) {
  /* 只有叶子节点存储数据 */
  if (level == 0) {
```

```
        page_cur_search_with_match(block, index, tuple, page_mode, &up_match,
                                   &up_bytes, &low_match, &low_bytes, cursor);
    }
}
```

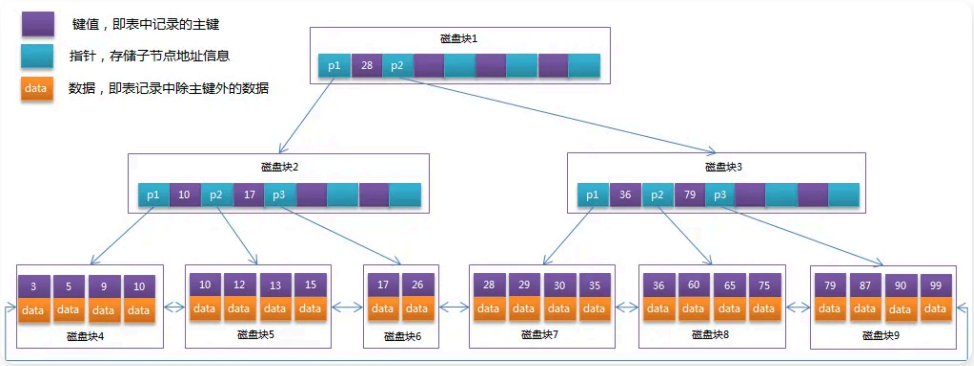
"原来 B+树通过叶子层双向链表，把离散的磁盘页变成了连续空间！"

整理好思绪，继续补充道：B+树通过以下创新实现质的飞跃：

1. **全数据叶子层**：所有数据仅存储在叶子节点，非叶节点仅作索引目录
2. **双向链表串联**：叶子节点通过指针形成有序链表，范围扫描时间复杂度从 O(logN)降为 O(1)。

在 B+Tree 中，所有数据记录节点都是按照键值大小顺序存放在同一层的叶子节点上，而非叶子节点上只存储 key 值信息，这样可以大大加大每个节点存储的 key 值数量，降低 B+Tree 的高度。

B+Tree 的非叶子节点只存储键值信息，假设每个磁盘块能存储 4 个键值及指针信息，则变成 B+Tree 后其结构如下图所示：



通常在 B+Tree 上有两个头指针，一个指向根节点，另一个指向关键字最小的叶子节点，而且所有叶子节点（即数据节点）之间是一种链式环结构。

因此可以对 B+Tree 进行两种查找运算：一种是对于主键的范围查找和分页查找，另一种是从根节点开始，进行随机查找。

MysQL 之父眼睛冒光，看着我惊呆了！！恨不得叫我一声大师。

西湖论剑，单挑首席

一月后，全球数据库峰会在西子湖畔召开。林渊抱着一台 IBM 服务器走上讲台："给我 30 秒，让各位见见‘未来索引’！”

实时 PK 表演：

```
-- 场景：1亿订单数据查询
-- 传统B树（甲骨文）
SELECT * FROM orders WHERE id BETWEEN 100000 AND 200000;
-- 耗时12.8秒

-- B+树（林渊魔改版）
SELECT /*+ BPLUS_SCAN */ * FROM orders BETWEEN 100000 AND 200000;
-- 耗时0.3秒
```

名场面台词：

"诸位，这不是优化，是维度的碾压！

B+树把磁盘的物理运动，变成了内存的闪电舞蹈！"——当日登上《程序员》杂志封面。三月后，林渊成立"深空科技"，发布"伏羲 B+引擎"。

美国商务部紧急会议："绝不能让中国掌控数据库心脏！"

最后福利

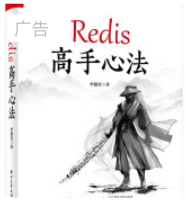
最后，宣传下自己的新书**《Redis 高手心法》**，上市后得到了许多读者的较好口碑评价，而且上过京东榜单！原创不易，希望大家多多支持，谢谢啦。

本书基于 Redis 7.0 版本，将复杂的概念与实际案例相结合，以简洁、诙谐、幽默的方式揭示了Redis的精髓。

从 Redis 的第一人称视角出发，拟人故事化方式和诙谐幽默的言语与各路“神仙”对话，配合 158 张图，由浅入深循序渐进的讲解 Redis 的数据结构实现原理、开发技巧、运维技术和高阶使用，让人轻松愉快地学习。

以下是读者的好评：

点击下方卡片即可购买



Redis 高手心法

京东配送

¥83.75

购买

京东

○ 往期推荐 ○

Kafka 4.0 发布：KRaft 替代 Zookeeper、新一代重平衡协议、点对点消息模型、移除旧协议 API

38 张图详解 Redis：核心架构、发布订阅机制、9大数据类型底层原理、RDB和AOF 持久化、高可...

拼多多二面：高并发场景扣减商品库存如何防止超卖？

云原生时代的JVM调优：从被K8s暴打到优雅躺平

高并发系统必看！G1如何让亿级JVM吞吐量提升300%？

性能提升300%！JVM分配优化三板斧，JVM 的内存区域划分、对象内存布局、百万 QPS 优化实践

从 12s 到 200ms，MySQL 两千万订单数据 6 种深度分页优化全解析

码哥跳动

《Redis 高手心法》作者，后端架构师，精通Java与Go，宗旨是拥抱技术和对象，面向人...

239篇原创内容

公众号

MySQL · 目录

上一篇

重生之MySQL 索引失效六大陷阱

下一篇

一文搞懂 MySQL InnoDB架构 Buffer Pool、Change Buffer、自适应哈希索引、