

图解 6 种「树」，你心中有数吗。。。>

小林coding 今天

以下文章来源于后端技术学堂，作者LemonCoder



后端技术学堂

一线互联网工程师，分享后端技术学习Linux/C++/Python/Go和后端组件、架构等方...>

数据结构这门课程是计算机相关专业的基础课，数据结构指的是数据在计算机中的存储、组织方式。

我们在学习数据结构时候，会遇到各种各样的基础数据结构，比如堆栈、队列、数组、链表、树...这些基本的数据结构类型有各自的特点，不同数据结构适用于解决不同场景下的问题。

树形结构相比数组、链表、堆栈这些数据结构来说，稍微复杂一点点，但树形结构可以用于解决很多实际问题，因为现实世界事物之间的关系往往不是线性关联的，而「树」恰好适合描述这种非线性关系。

今天就带大家一起学习下，数据结构中的各种「树」，这也是面试中经常考察的内容，手撕二叉树是常规套路，对候选人也很有区分度，学完这篇文章，相信大家都会心中有「树」了。

从树说起

什么是树？现实中的树大家都见过，在数据结构中也有树，此树非彼树，不过数据结构的树和现实中的树在形态上确实有点相像。

树是非线性的数据结构，用来模拟具有树状结构性质的数据集合，它是由 n 个有限节点组成的具有层次关系的集合。在数据结构中树是非线性数据结构，那我们先来了解下，什么是线性与非线性数据结构？

线性结构

「线性结构」是一个有序数据元素的集合。其中数据元素之间的关系是一对一的关系，即除了第一个和最后一个数据元素之外，其它数据元素都是首尾相接的，常用的线性结构有：线性表，栈，队列，双端队列，数组，串。



堆棧

可以想象，所谓的线性结构数据组织形式，就像一条线段一样笔直，元素之间首尾相接。比如现实中的火车进站、食堂打饭排队的队列。

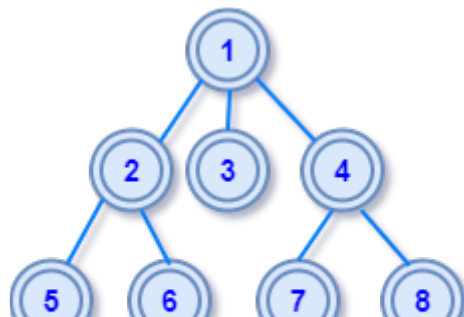
非线性结构

简而言之，线性结构的对立面就是「非线性结构」。

树

线性结构中节点是首位相接一对一关系，在树结构中节点之间不再是简单的一对一关系，而是较为复杂的一对多的关系，一个节点可以与多个节点发生关联，树是一种层次化的数据组织形式，树在现实中是可以找到例子的，比如现实中的族谱，亲戚之间的关系是层次关联的树形关系。

数据结构中的「树」的名字由来，是因为如果把节点之间的关系直观展示出来，由于长得和现实世界中的树很像，由此得名。如图：

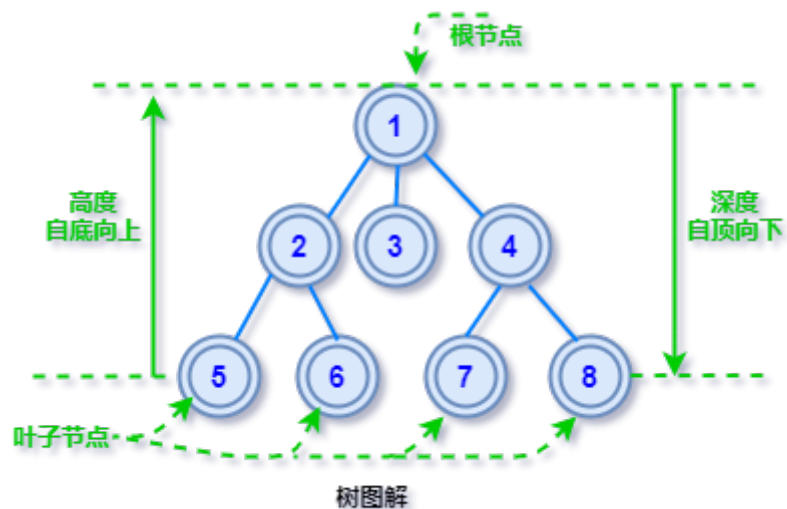




树的关键概念

人们对树形结构的研究比较深入，为了方便研究树的各种性质，抽象出了一些树相关的概念，以便清晰简介的描述一颗树。下面几个基础概念必须了解，否则你当你刷LeetCode树相关题目时候，或者面试官向你描述问题时，你会连题目都看不懂是什么意思。

先来上一个图解，下面的术语和概念对照着看，更容易理解。



什么是节点的度？

”

度很好理解，直观来说，数一下节点有几个分叉就说这个节点的度是多少。

“

什么是根节点？

”

在一颗树形结构中，最顶层的那个节点就是根节点了，所有的子节点都源自它发散开来。

“

什么是父节点？

”

树的父子关系和现实中很相似，若一个节点含有子节点，则这个节点称为其子节点的父节点。

“

什么是叶子节点？

”

直观来看叶子节点都位于树的最底层，就是没有分叉的节点，严格的定义是度为 0 的节点叫叶子节点。

“

什么是节点的高度？

”

高度是从叶子节点开始「自底向上」逐层累加的路径长度，树叶的高度为 0（有些书上也说是 0，不用纠结）

“

什么是节点的深度？

”

深度是从根节点开始「自顶向下」逐层累加的路径长度，根的深度为1（有些书上也说是 0，问题不大）

小技巧：高度和深度，一个从下往上数，一个从上往下数。

树的特点

树形数据结构，具有以下的特点：

- 。每个节点都只有有限个子节点或无子节点；

- 没有父节点的节点称为根节点；
- 每一个非根节点有且只有一个父节点；
- 除了根节点外，每个子节点可以分为多个不相交的子树；
- 树里面没有环路，意思就是从有一个节点出发，除非往返，否则不能回到起点。

二叉树

有了前面「树」的基础铺垫，二叉树是一种特殊的树，还记的上面我们学过「节点的度」吗？二叉树中每个节点的度不大于 2，即它的每个节点最多只有两个分支，通常称二叉树节点的左右两个分支为左右子树。

二叉树是很多其他树型结构的基础结构，比如下面要讲的 AVL 树、二叉查找树，他们都是由二叉树增加一些约束条件进化而来。

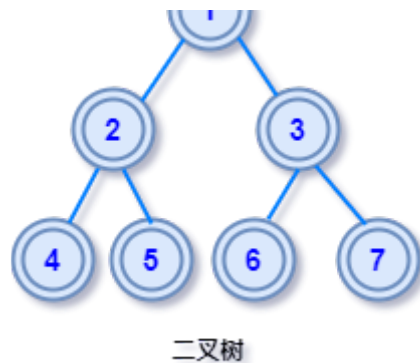
三种遍历方式

二叉树的遍历就是逐个访问二叉树节点的数据，常见的二叉树遍历方式有三种，分别是前中后序遍历，初学者分不清这几个顺序的差别。

「有个简单的记忆方式，这里的「前中后」都是对于根节点而言」。

先访问根节点后访问左右子树的遍历方式是前序遍历，先访问左右子树最后访问根节点的遍历方式是后序遍历，先访问左子树再访问根节点最后访问右子树的遍历方式是中序遍历，下面详细说明：





前序遍历

遍历顺序是根节点->左子树->右子树

遍历的得到的序列是： 1 2 4 5 3 6 7

中序遍历

遍历顺序是左子树->根节点->右子树

遍历的得到的序列是： 4 2 5 1 6 3 7

后序遍历

遍历顺序是左子树->右子树->根节点

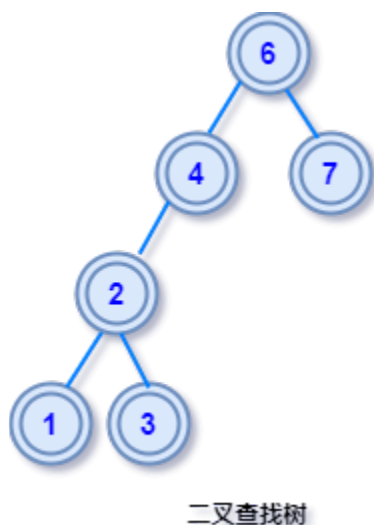
遍历的得到的序列是： 4 5 2 6 7 3 1

二叉查找樹

由于最基础的二叉树节点是无序的，想象一下如果在二叉树中查找一个数据，最坏情况可能要遍历整个二叉树，这样的查找效率是非常低下的。

由于基础二叉树不利于数据的查找和插入，因此我们有必要对二叉树中的数据进行排序，所以就有了「二叉查找树」，可以说这种树是为了查找而生的二叉树，有时也称它为「二叉排序树」，都是同一种结构，只是换了个叫法。

查找二叉树理解了也不难，简单来说就是二叉树上所有节点的，左子树上的节点都小于根节点，右子树上所有节点的值都大于根节点。



这样的结构设计，使得查找目标节点非常方便，可以通过关键字和当前节点的对比，很快就能知道目标节点在该节点的左子树还是右子树上，方便在树中搜索目标节点。

如果对排序二叉树执行中序遍历，因为中序遍历的顺序是：左子树->根节点->右子树，最终可以得到一个节点值的有序列表。

举个栗子：对上图的排序二叉树执行中序遍历，我们可以得到一个有序序列：**1 2 3 4 5 6 7**

查询效率

二叉查找树的查询复杂度取决于目标节点的深度，因此当节点的深度比较大时，最坏的查询效率是 $O(n)$ ，其中 n 是树中的节点个数。

实际应用中有许多改进版的二叉查找树，目的是尽可能使得每个节点的深度不要过深，从而提高查询效率。比如AVL树和红黑树，可以将最坏效率降低至 $O(\log n)$ ，下面我们就来看下这两种改进的二叉树。

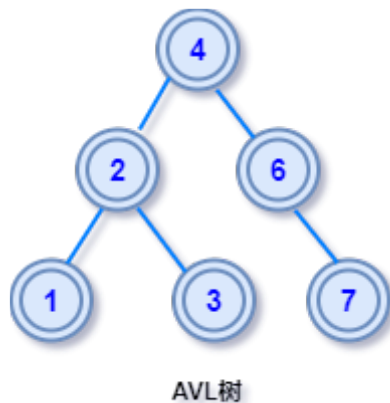
AVL树

AVL 也叫平衡二叉查找树。AVL 这个名字的由来，是它的两个发明者G. M. Adelson-Velsky 和 Evgenii Landis 的缩写，AVL最初是他们两人在1962 年的论文「An algorithm for the organization of information」中提出来一种数据结构。

定义

AVL 树是一种平衡二叉查找树，二叉查找树我们已经知道，那平衡是什么意思呢？

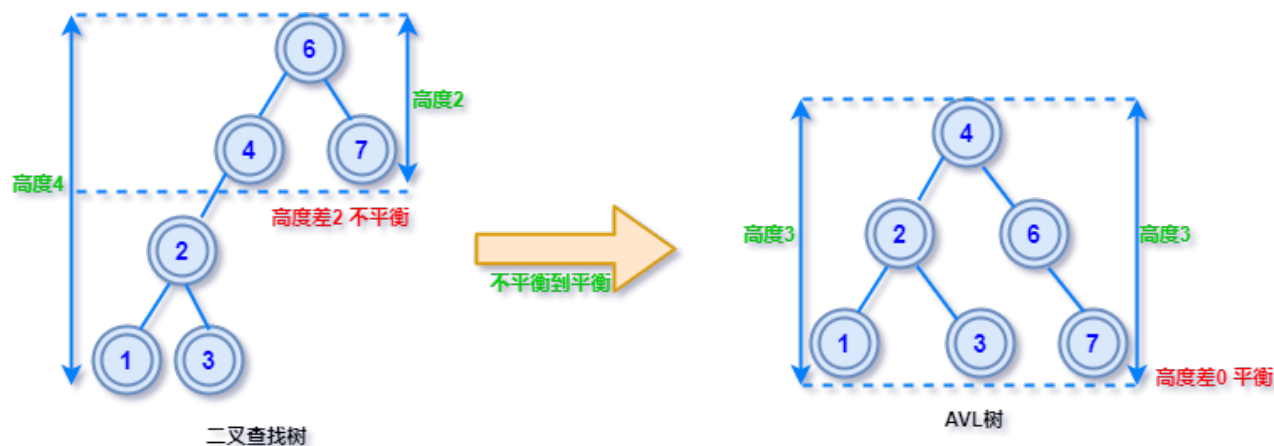
我们举个天平的例子，天平两端的重量要差不多才能平衡，否则就会出现向一边倾斜的情况。把这个概念迁移到二叉树中来，根节点看作是天平的中点，左子树的高度放在天平左边，右子树的高度放在天平右边，当左右子树的高度相差「不是特别多」，称为是平衡的二叉树。



AVL树有更严格的定义：在二叉查找树中，任一节点对应的两棵子树的最大高度差为 1，这样的二叉查找树称为平衡二叉树。其中左右子树的高度差也有个专业的叫法：平衡因子。

AVL树的旋转

一旦由于插入或删除导致左右子树的高度差大于1，此时就需要旋转某些节点调整树高度，使其再次达到平衡状态，这个过程称为旋转再平衡。



保持树平衡的目的是可以控制查找、插入和删除在平均和最坏情况下的时间复杂度都是 $O(\log n)$ ，相比普通二叉树最坏情况的时间复杂度是 $O(n)$ ，AVL树把最坏情况的复杂度控制在可接受范围，非常合适对算法执行时间敏感类的应用。

B 树

B树是鲁道夫·拜尔 (Rudolf Bayer) 1972年在波音研究实验室 (Boeing Research Labs) 工作时发明的，关于B树名字的由来至今是个未解之谜，有人猜是Bayer的首字母，也有人说是波音实验室 (Boeing Research Labs) 的Boeing首字母缩写，虽然B树这个名字来源扑朔迷离，我们心里也没点B树，但不影响今天我们来学习它。

定义

一个 m 阶的B树是一个有以下属性的树

1. 每一个节点最多有 m 个子节点
2. 每一个非叶子节点（除根节点）最少有 $\lceil m/2 \rceil$ 个子节点， $\lceil m/2 \rceil$ 表示向上取整。
3. 如果根节点不是叶子节点，那么它至少有两个子节点
4. 有 k 个子节点的非叶子节点拥有 $k - 1$ 个键
5. 所有的叶子节点都在同一层

如果之前不了解，相信第一眼看完定义肯定是蒙圈，不过多看几遍好好理解一下就好了，画个图例，对照着看看：

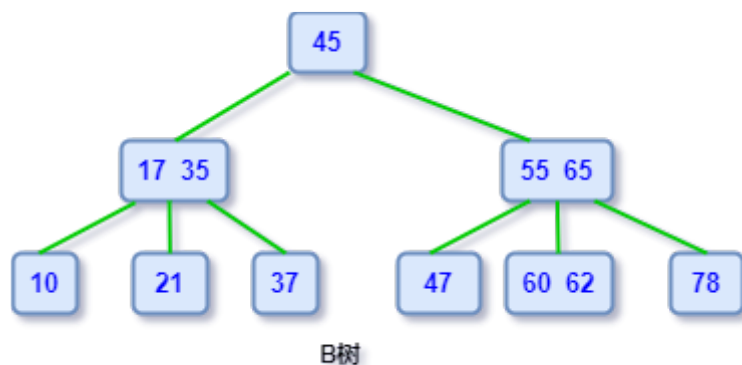


图3

特点

- B树是所有节点的平衡因子均等于0的多路查找树（AVL树是平衡因子不大于1的二路查找树）。

- B 树节点可以保存多个数据，使得 B 树可以不用像 AVL 树那样为了保持平衡频繁的旋转节点。
- B树的多路的特性，降低了树的高度，所以B树相比于平衡二叉树显得矮胖很多。
- B树非常适合保存在磁盘中的数据读取，因为每次读取都会有一次磁盘IO，高度降低减少了磁盘IO的次数。

B树常用于实现数据库索引，典型的实现，MongoDB索引用B树实现，MySQL的InnoDB 存储引擎用B+树存放索引。

说到B树不得不提起它的好兄弟B+树，不过这里不展开细说，只需知道，B+树是对B树的改进，数据都放在叶子节点，非叶子节点只存数据索引。

红黑树

红黑树也是一种特殊的「二叉查找树」。

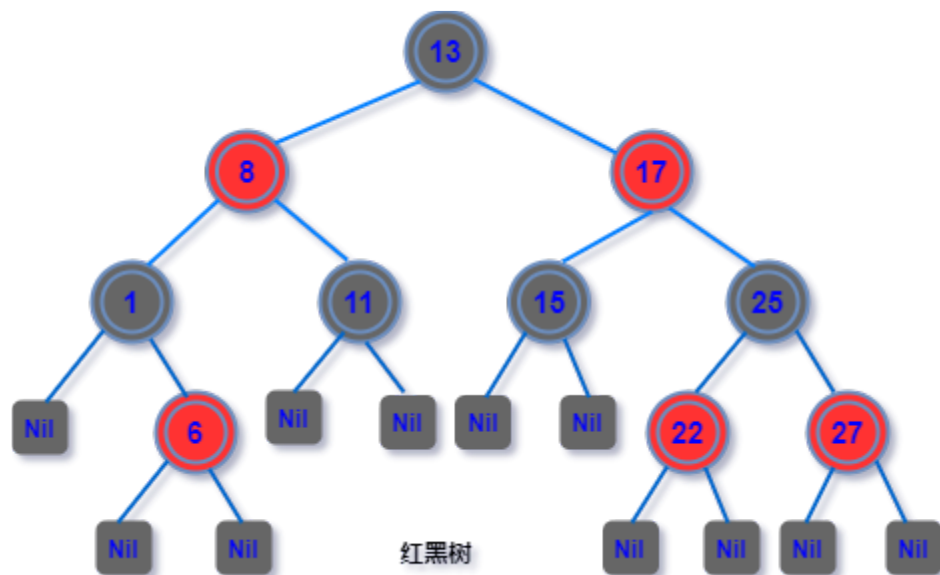
到目前为止我们学习的 AVL 树和即将学习的红黑树都是二叉查找树的变体，可见二叉查找树真的是非常重要基础二叉树，如果忘了它的定义可以先回头看看。

特点

红黑树中每个结点都被标记了红黑属性，红黑树除了有普通的「二叉查找树」特性之外，还有以下的特征：

1. 节点是红色或黑色。
2. 根是黑色。
3. 所有叶子都是黑色（叶子是NIL节点）。
4. 每个红色节点必须有两个黑色的子节点。（从每个叶子到根的所有路径上不能有两个连续的红色节点。）
5. 从任一节点到其每个叶子的所有简单路径都包含相同数目的黑色节点。

这些性质有兴趣可以自行研究，不过，现在你只需要知道，这些约束确保了红黑树的关键特性：从根到叶子的最长的可能路径不多于最短的可能路径的两倍长。



红黑树

而节点的路径长度决定着对节点的查询效率，这样我们确保了，最坏情况下的查找、插入、删除操作的时间复杂度不超过 $O(\log n)$ ，并且有较高的插入和删除效


率。

应用场景

红黑树在实际应用中比较广泛，有很多已经落地的实践，比如学习C++的同学都知道会接触到 STL 标准库，而STL容器中的map、set、multiset、multimap 底层实现都是基于红黑树。

再比如，Linux内核中也有红黑树的实现，Linux系统在实现EXT3文件系统、虚拟内存管理系统，都有使用到红黑树这种数据结构。

Linux内核中的红黑树定义在内核文件如下的位置：



```
头文件: linux-version/include/linux/rbtree.h  
源文件: linux-version/lib/rbtree.c  
其中, linux-version表示具体的linux版本号
```

如果找不到，可以 `find / -name rbtree.h` 搜索一下即可，有兴趣可以打开文件看下具体实现。

红黑树 VS 平衡二叉树 (AVL树)

- 插入和删除操作，一般认为红黑树的删除和插入会比 AVL 树更快。因为，红黑树不像 AVL 树那样严格的要求平衡因子小于等于1，这就减少了为了达到平衡而进行的旋转操作次数，可以说是牺牲严格平衡性来换取更快的插入和删除时间。
- 红黑树不要求有不严格的平衡性控制，但是红黑树的特点，使得任何不平衡都会在三次旋转之内解决。而 AVL 树如果不平衡，并不会控制旋转操作次数，旋转直到平衡为止。
- 查找操作，AVL树的效率更高。因为 AVL 树设计比红黑树更加平衡，不会出现平衡因子超过 1 的情况，减少了树的平均搜索长度。

Trie 树 (前缀树或字典树)

Trie来源于单词 retrieve (检索)，Trie树也称为前缀树或字典树。利用字符串前缀来查找指定的字符串，缩短查找时间提高查询效率，主要用于字符串的快速查找和匹配。

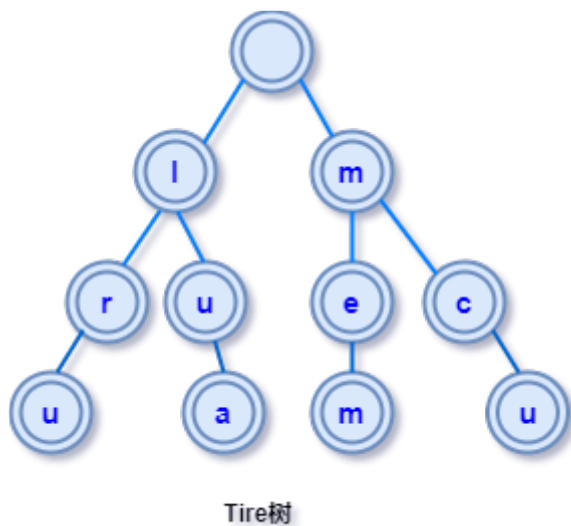
为什么要称其为字典树呢？因为Trie树的功能就像字典一样，想象一下查英文字典，我们会根据首字母找到对应的页码，接着根据第二、第三...个单词，逐步查找目标单词，Trie树的组织思想和字典组织很像，字典树由此得名。

定义

Trie的核心思想是空间换时间，有 3 个基本性质：

1. 根节点不包含字符，除根节点外每一个节点都只包含一个字符。
2. 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。
3. 每个节点的所有子节点包含的字符都不相同。

比如对单词序列 `lru, lua, mem, mcu` 建立Trie树如下：



Trie树建立和查询是可以同步进行的，可以在还没建立出完成的 Trie 树之前就找到目标数据，而如果用 Hash 表等结构存储是需要先建立完成表才能开始查询，这也是 Trie 树查询速度快的原因。

应用

Trie树还用于搜索引擎的关键词提示功能。比如当你在搜索框中输入检索单词的开头几个字，搜索引擎就可以自动联想匹配到可能的词组出来，这正是Trie树的最直接应用。



这种结构在海量数据查询上很有优势，因为不必为了找到目标数据遍历整个数据集，只需按前缀遍历匹配的路径即可找到目标数据。

因此，Trie树还可用于解决类似以下的面试题：

“

给你100000个长度不超过10的单词。对于每一个单词，我们要判断他出没出现过，如果出现了，求第一次出现在第几个位置。

“

有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M，求频数最高的100个词

”

“

1000万字符串，其中有些是重复的，需要把重复的全部去掉，保留没有重复的字符串，请问怎么设计和实现？

”

“

一個文本文件，大約有一萬行，每行一個詞，要求統計出其中最頻繁出現的前10個詞，請給出思想，給出時間複雜度分析。

”

總結一下

樹形數據結構有許多變種，這篇文章我們從樹開始，把幾種常見樹形數據結構學習了一遍，包括二叉樹、二叉查找樹（二叉搜索樹）、AVL樹、紅黑樹、B樹、Trie樹。

文章構思的時候想聊聊數據結構中的樹，沒想到步子跨的有點大，大到不知從何說起，因為數據結構中各種樹的變體非常多，一篇文章實難細數，篇幅有限，也只能說是淺嚐輒止，作為樹形數據結構入門，如果要深入的學習，每個知識點還能寫出一篇文章，比如B+ 樹原理及其在數據庫索引中的應用、紅黑樹的詳細分析等等，這次檸檬只是粗略帶大家走一遍。

在後端開發中，數據結構與算法是後端程序員的基本素養，除了基礎架構部的後端開發同學，雖然我們平常不會經常造輪子，但是掌握基本的數據結構與算法仍然是非常有必要，面試也對相關能力有要求。

回顧往期文章，數據結構的內容寫的比較少，如果大家有興趣，檸檬會再多寫一些相關主題的文章！

感謝各位的閱讀，文章的目的是分享對知識的理解，技術類文章我都會反复求證以求最大程度保證準確性，若文中出現明顯紕漏也歡迎指出，我們一起在探討中學習。



圖解計算機基礎
认准**小林coding**

每一張圖都包含小林的認真
只為帮助大家能更好的理解

① 关注公众号回复「**网络**」
送你原创 300 页的图解网络

② 关注公众号回复「**加群**」
拉你进百人技术交流群

扫一扫，关注「小林coding」公众号

哈嘍，我是小林，就愛圖解計算機基礎，如果覺得文章對你有幫助，歡迎分享給你的朋友，也給小林點個「在看」，這對小林非常重要，謝謝你們，給各位小姐姐小哥哥們抱拳了，我們下次見！

推薦閱讀

學完計組後，我馬上在「我的世界」造了台顯示器，你敢信？

10 張圖打開CPU 緩存一致性的大門

