链表

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.08

目录

01

链表是什么

TEACHING
COURSEWARE

TEACH

链表是什么

链表是一种线性数据结构，由一系列 “节点” 通过 “指针” 连接而成，用于存储有序的数据集合。

为什么要有链表？

链表是什么

数组：数组是最基础的线性结构，但它有两个明显的局限性：
1. 内存空间“刚性” 数组需要连续的内存空间，并且在创建时必须确定大小（比如int arr[100]）。 - 如果空间开小了，数据存不下，需要重新申请更大的连续空间，把原有数据“搬过去”，这个过程很耗时（比如从arr[100]扩容到arr[200]，要复制100个元素）。
如果空间开大了，又会浪费内存（比如只存10个数据，却占用了100个位置的空间）。这在数据量不确定的场景下非常不方便。
2. 增删元素，数组中插入或删除元素时，需要移动大量元素来“腾位置”或“补空位”。
比如在数组[1,2,3,4,5]的第2个位置插入6，需要把2,3,4,5都往后移一位，变成[1,6,2,3,4,5]，移动次数随数据量增加而增加（时间复杂度O(n)）。
删除元素同理，比如删除3，需要把4,5往前移，同样要移动多个元素。

链表是什么

1. 不依赖连续内存，空间更灵活链表的节点可以分散存储在内存的任何位置，只需通过指针“串”起来。 不需要预先确定大小，数据多了就新增节点，数据少了就删除节点。
2. 增删元素更高效 只要找到目标位置，链表增删节点只需修改指针指向，不用移动其他节点： 比如在链表1→2→3→4中插入5到2和3之间，只需让2的指针指向5，5的指针指向3（两步操作），1和4等其他节点完全不用动。

什么时候特别需要链表？
数据量不确定，需要频繁动态增删；
不需要频繁“随机访问”（即通过下标快速定位元素），更多是“顺序遍历”；
希望减少内存浪费或避免扩容时的大量数据搬迁。

02

链表概念

TEACHING
COURSEWARE

TEACH

链表概念

链表是一种线性数据结构，由若干个称为 “节点”（Node）的基本单元通过指针 依次连接而成，用于存储具有逻辑顺序的元素集合。其核心特征包括：

1. 节点结构

每个节点包含两部分信息：

数据域（Data Field）：存储当前节点的具体数据（如整数、字符串、对象等）。

指针域（Pointer Field）：存储指向链表中下一个节点（或前驱节点，双向链表中） 的地址，用于建立节点间的逻辑关联。

2. 线性逻辑与非连续存储

从逻辑上看，链表的节点按顺序依次排列，形成 “线性序列”，符合线性结构 “除首尾元素外，每个元素有唯一前驱和后继” 的特征。

3. 动态性

链表的长度可在程序运行时动态调整，无需预先指定固定大小。插入或删除节点时，仅需修改相关节点的指针域，无需移动其他元素。

链表概念

1. 表头与表尾

链表的第一个节点称为头节点（Head Node），其指针域指向第一个有效节点（或为空）；通常用一个 “头指针” 指向头节点，作为链表的入口。

链表的最后一个节点称为尾节点（Tail Node），其指针域为空（表示没有后续节点）。

2. 衍生结构

基于上述定义可扩展出多种变体，例如：

双向链表：每个节点包含两个指针域，分别指向前驱节点和后继节点。

循环链表：尾节点的指针域指向头节点（或第一个有效节点），形成闭合回路。

03

链表设计

TEACHING
COURSEWARE

TEACH

链表设计

1. 插入节点（以单链表为例）

场景：在节点 A 和节点 B 之间插入节点 C。

步骤：

先让 C 的指针指向 B（C→next = B）；

再让 A 的指针指向 C（A→next = C）。

关键：顺序不能反！如果先改 A 的指针，会丢失 B 的地址。

2. 删除节点（以单链表为例）

场景：删除节点 B（假设已知其前一个节点 A）。

步骤：

让 A 的指针直接指向 B 的下一个节点（A→next = B→next）。

链表设计

特性	数组	链表
存储方式	连续的内存空间	非连续的内存空间，靠指针连接
访问元素	通过下标直接访问（O（1））	必须从表头依次遍历（O（n））
插入 / 删除	需移动大量元素（O（n））	只需修改指针（O（1），找到位置后）
内存灵活性	固定大小，初始化时确定	动态大小，可随时增减节点

已经找到了要删除的节点及其前驱节点

04

总结

TEACHING
COURSEWARE

TEACH