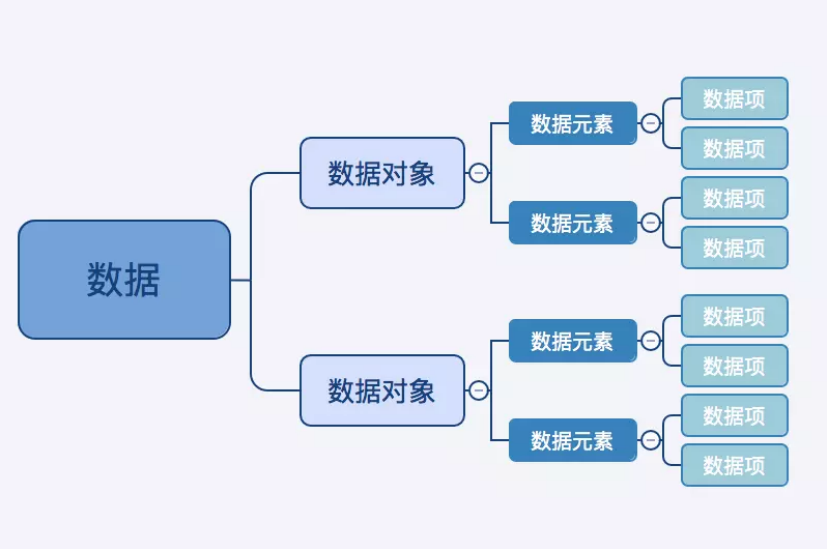
数据结构概念

结构：简单的理解就是关系，比如分子结构，就是说组成分子的原子之间的排列方式。严格点说， 结构是指各个组成部分相互搭配和排列的方间不是独立的，而是存在特定的关系，我们将这些关系称为结构 。

数据结构：相互之间存在一种或多种特定 关系的数据元素的集合

数据的组成结构



数据结构分类

同样是结构，从不同的角度来讨论，会有不同的分类可以分为逻辑结构和物理结构

逻辑结构

是指数据对象中数据元素之间的相互关系，可以分类四类

（1）集合结构：集合结构中的元素关系，除了同属于一个集合这个关系以外，再无其他关系。

（2）线性结构：线性结构中，元素间的关系就是一对一，顾名思义，一条线性的结构。

（3）树形结构：树形结构中，元素间的关系就是一对多，一颗大叔，伸展出的枝叶，也是类金字塔形。

（4）图形结构：图形结构中，元素间的关系就是多对多，类蛛网形。

物理结构

也叫做存储结构，是指数据的逻辑结构在计算机中的存储形式。

数据是数据元素的集合，那么根据物理结构的定义，实际上就是如何把数据元素存储到计算机的存储器中。存储器主要是针对内存而言的，像
组织通常用文件结构来描述

物理结构就是讲究内存的存储方式也分两种：

（1）顺序存储结构：是把数据元素存放在地址连续存储单元里，其数据间的逻辑关系和物理关系是一致的

（2）链式存储结构：既然有这种结构就是跟顺序存储结构有了对比，那就是其中逻辑关系和物理关系没有多大的关系因为其中的数据元素
活；

> 链式存储结构是把数据元素存放在任意的存储单元里，这组存储单元可以是连续的也可以是不连续的。
>
> 这样的话链式存储结构的数据元素存储关系并不能反映其逻辑关系，因此需要用一个指针存放数据元素的地址，这样子通过地址就可

算法

是解决特定问题步骤的描述,在计算机中表现为指令的有限序列,每条指令表示一个或多个操作

算法效率的度量方法

- 事后统计方法 —— 设计测试数据,在计算机上运行算法,看时间(不采用,因为硬件差距大)
- 事前分析估算方法- —— 依据统计方法对算法进行估算,消耗时间的基本操作的执行次数(输入规模)

两种算法

求出 1加到100的和

算法1：

```
int i,sum=0;
int n=100;
for (i = 1; i < = n; i++) {
    sum = sum + i ;
}
printf ( s u m ) ;
```

算法2：

```
int 1, sum = 0;
int n = 100;
sum = (1 + n) * n / 2;
printf (sum);
```

第一种计算次数会随着N增加而增加，而第二种算法不论N是多少，计算次数都不会变化，如果N等于100000000000，那
结果，而第二种算法计算需要的时间没变化，这种随着 N变化而 对计算时间的变化叫做“函数的渐近增长”

时间复杂度

时间频率 ： 一个算法花费的时间与算法中语句的执行次数成正比例，哪个算法中语句执行次数多，它花费时间就多。一个
频度。记为T(n)

时间复杂度： 在刚才提到的时间频度中，n称为问题的规模，当n不断变化时，时间频度T(n)也会不断变化。但有时我们想
此，我们引入时间复杂度概念。 一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数，用T(n)表示，若有
无穷大时，T(n)/f(n)的极限值为不等于零的常数，则称f(n)是T(n)的同数量级函数。记作T(n)=O(f(n)),称O(f(n)) 为算法的渐

推导大 O 阶方法

那么如何分析一个算法的时间复杂度呢?即如何推导大 0 阶呢?我们给出了下面的推导方法，基本上，这也就是总结前面我们

```
推导大 O 阶 ：
1.用常数 1 取代运行时间中的所有加法常数
2.在修改后的运行次数函数中，只保留最高阶项 。
3.如果最高阶项存在且不是 1 ，则去除与这个项相乘的常敢 。 得到的结果就是大 O 阶
```

常数阶 —— O(1)

首先顺序结构的时间复杂度

```
int 1, sum = 0;          # 执行一次
int n = 100;             # 执行一次
sum = (1 + n) * n / 2;   # 执行一次
```

这个算法的运行次数函数是 f(n) =30 根据我们推导大 0 阶的方法，第一步就是把常数项 3 改为 1。在保留 最高阶项时发现
算法的时间复杂度为 0(1)

```
int 1, sum = 0;          # 执行一次
int n = 100;             # 执行一次
sum = (1 + n) * n / 2;   # 执行一次
sum = (1 + n) * n / 2;   # 执行一次
sum = (1 + n) * n / 2;   # 执行一次
sum = (1 + n) * n / 2;   # 执行一次
sum = (1 + n) * n / 2;   # 执行一次
```

事实上无论 n 为多少，上面的两段代码就是 3 次和 7 次执行的差异。这种与问 题的大小无关 (n 的多少)，执行时间恒定的间复杂 度，又叫常数阶。

注意 : 不管这个常数是多少，我们都记作 O(1) ，而不能是 0(3)、 0(12)等其他任 何数字

线性阶 —— O(n)

下面这段代码，它的循环的时间复杂度为 O(n)， 因为循环体中的代码须要执行 n 次

```
int i;
for (i = 0; i < n; i++){

}
```

对数阶 —— O(logN)

```
int count = 1;
while (count < n){
    count = count * 2;
}
```

由于每次count乘以2之后，就距离n更近了一分。也就是说，有多少个2相乘后大于n，则会退出循环。由2x=n得到x=log2r O(logn)

平方阶 —— $O(n^2)$

```
int i,j;
for (int i = 0 ; i < n ; I++){
    for (j = 0 ; j < n; j++){

    }
}
```

对于外层循环时间复杂度为 N， 对于内层循环时间复杂度也为N，所以这段代码的时间复杂度为 n^2