

数据结构

第 1 周 – 绪论、数据与基本数据类型

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

关于主讲教师

- **Koczka Ferenc** 博士，助理教授
- 埃斯特哈齐·卡罗伊大学（Eszterházy Károly Egyetem），匈牙利埃格尔
- 电子邮件：koczka.ferenc@uni-eszterhazy.hu
- 答疑：课后，或通过电子邮件预约

“先解决问题，再写代码。” – John Johnson

课程安排

周	主题	周	主题
1	绪论、基本数据类型	9	动态内存、链表
2	效率、查找	10	带哨兵的链表、双向链表、循环链表
3	数组	11	链栈与链队列、哈希表
4	字符串	12	递归
5	排序 I（简单排序）	13	排序 II（归并排序、快速排序）
6	集合	14	树、二叉树、遍历
7	栈	15	二叉搜索树、堆、总结
8	队列	16	期末笔试和上机考试

每周：2 学时讲课 + 2 学时实践课（Java 和 Go）

考核方式与工具

考核（第 16 周）

- **期末笔试**（理论，60 分钟）
判断题、选择题、代码阅读、伪代码填空
- **上机考试**（在计算机上）
用 Java **或** Go 实现一个数据结构

工具

- **Java** – JDK 17 或更新版本
- **Go** – 1.23 或更新版本
- 编辑器：VS Code、IntelliJ IDEA 或 Eclipse
- 示例程序： `code/java/wXX` 、 `code/go/wXX`

为什么要学习数据结构？

算法 + 数据结构 = 程序

– Niklaus Wirth(1976)

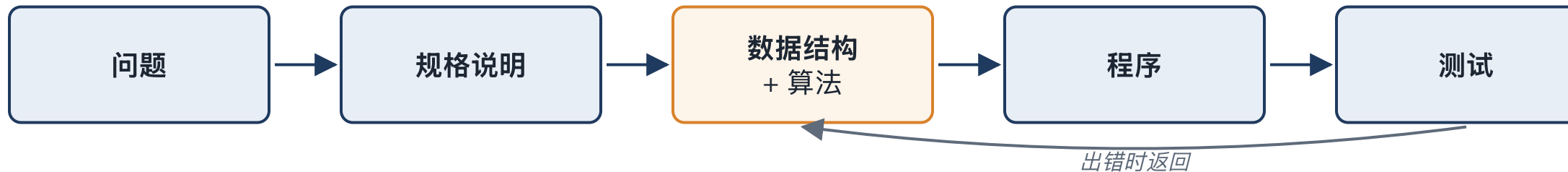
- 同一个任务，用**好的**数据结构很快，用**差的**数据结构可能慢得无法使用
- 每一种编程语言都建立在这些基础之上（`ArrayList`、`HashMap`、Go 的 `slice`、`map` ...）
- 求职面试中最常见的话题之一
- 后续课程（数据库、操作系统、人工智能）都以这些知识为前提

目标：掌握基本的数据结构、它们的操作及其效率 – 并且能够**动手实现**它们。



问题求解与算法

问题求解的步骤



- **规格说明**：输入是什么，输出是什么，必须满足哪些条件？
- **数据结构 + 算法**：如何存储数据，按什么步骤处理数据？
- 本课程讲的就是中间这一步

算法的概念

算法：由有限个定义明确的步骤组成、用于解决某个问题的过程。

算法的特性：

- **有穷性** – 在有限步之后结束
- **确定性** – 每一步都有精确的定义
- **有输入** – 零个或多个数据
- **有输出** – 至少一个结果
- **可行性** – 每一步都能够实际执行
- **通用性** – 解决一类问题，而不只是某一个情况

算法的描述方法

工具	优点	缺点
自然语言	人人都能看懂	不精确、冗长
流程图	直观	问题较大时难以阅读
伪代码	精确、与语言无关	需要学习记号
程序代码	可以执行	有语言相关的细节

本课程：讲课和笔试中使用**伪代码**，实践课中使用 **Java 和 Go**。

伪代码记号

记号	含义
<code>x ← expression</code>	赋值
<code>If condition then ... else ... End if</code>	选择（分支）
<code>While condition ... End while</code>	当型循环（先判断）
<code>For i ← 1 to n ... End for</code>	计数循环
<code>Function Name(parameters) ... Return value ... End function</code>	函数
<code>Procedure Name(parameters) ... End procedure</code>	过程（没有返回值）
<code>A[i]</code>	数组的第 i 个元素 – 下标从 1 开始
<code>and</code> 、 <code>or</code> 、 <code>not</code> 、 <code>div</code> 、 <code>mod</code>	逻辑运算 · 整除、取余

在 Java 和 Go 中数组下标从 **0** 开始 – 写代码时一定要注意！

示例：最大公约数（欧几里得算法）

伪代码

```
Function GCD(a, b)
  While b  $\neq$  0
    m  $\leftarrow$  a mod b
    a  $\leftarrow$  b
    b  $\leftarrow$  m
  End while
  Return a
End function
```

a	b	a mod b
1071	462	147
462	147	21
147	21	0
21	0	

Java

```
static int gcd(int a, int b) {
    while (b != 0) {
        int m = a % b;
        a = b;
        b = m;
    }
    return a;
}
```

Go

```
func gcd(a, b int) int {
    for b != 0 {
        a, b = b, a%b
    }
    return a
}
```



数据与数据类型

数据与信息

- **数据**：对现实事物某种属性的记录，是可以被处理的形式
例如 37.2 、 "Wuhan" 、 true
- **信息**：数据在特定语境中的含义
例如“病人的体温是 37.2 °C”
- 计算机处理的是**数据** – 含义由程序（以及程序员）赋予

数据的属性

- **名字**（标识符） – 我们如何引用它： temperature
- **类型** – 它能取哪些值，能对它做什么
- **值** – 它当前的内容
- **地址** – 它在内存中的位置

数据类型

数据类型规定了三件事：

1. **值的集合** – 可以取哪些值
2. **运算** – 可以对这些值做什么
3. **表示** – 机器如何存储它（多少位、什么编码）

示例 – Java 的 `int`：

取值：-2 147 483 648 ... 2 147 483 647 · 运算：+ - * / % == < ... · 表示：32 位，补码

分类

- **基本（简单）类型**：不能再分解 – 布尔、字符、整数、实数、指针
- **构造类型（数据结构）**：由基本数据组成 – 数组、记录、链表 ...

Java 和 Go 中的基本类型

种类	Java	Go	大小
布尔	boolean	bool	–
字符	char (UTF-16 代码单元)	rune (= int32 ,Unicode 码点)	16 / 32 位
字节	byte (有符号)	byte (= uint8 , 无符号)	8 位
整数	short 、 int 、 long	int8 、 int16 、 int32 、 int64 、 int	16 / 32 / 64 位
无符号整数	–	uint8 ... uint64 、 uint	8 ... 64 位
实数	float 、 double	float32 、 float64	32 / 64 位
指针 / 引用	引用 (指向对象)	*T 指针	64 位

Go 中 int 的大小取决于机器（如今几乎总是 64 位）；Java 中 int 永远是 32 位。

整数的表示

5 =	0	0	0	0	0	1	0	1
-5 =	1	1	1	1	1	0	1	1
127 =	0	1	1	1	1	1	1	1
-128 =	1	0	0	0	0	0	0	0

符号位

byte: 8 位

范围: $-128 \dots 127$

取负: 按位取反 + 1

- 用 n 位可以表示 2^n 个不同的值
- 有符号整数 (补码): $-2^{n-1} \dots 2^{n-1} - 1$
- Java 的 `int` (32 位): $-2\ 147\ 483\ 648 \dots 2\ 147\ 483\ 647$

溢出

如果运算结果放不进该类型，值就会“**回绕**” – 而且没有任何错误提示！

Java

```
int max = Integer.MAX_VALUE;  
System.out.println(max + 1);  
// -2147483648  
  
byte b = 127;  
b++;           // -128
```

Go

```
var max int32 = math.MaxInt32  
max++  
fmt.Println(max)  
// -2147483648  
  
var b int8 = 127  
b++           // -128
```

一定要选择能放下最大预期值的类型（例如 `long` / `int64`）。

实数：浮点表示

- 形式：符号 × 尾数 × $2^{\text{指数}}$ (IEEE 754 标准)
- float / float32：约 7 位有效数字 · double / float64：约 15–16 位有效数字
- 大多数十进制小数（例如 0.1）在二进制中**无法精确表示**

```
double s = 0.1 + 0.2;  
System.out.println(s);  
// 0.30000000000000004  
System.out.println(s == 0.3); // false
```

解决办法：带容差比较

```
final double EPS = 1e-9;  
Math.abs(s - 0.3) < EPS // true
```

金额计算中使用整数（例如以“分”为单位）或 `BigDecimal` 类型。

整数除法与类型转换

- 两个整数相除的商是**整数**： `7 / 2 == 3`，余数： `7 % 2 == 1`
- 只要有一个操作数是实数，结果就是实数： `7 / 2.0 == 3.5`

```
double f = 9 / 5 * celsius + 32;    // 错误: 9 / 5 == 1
double f = 9.0 / 5 * celsius + 32;   // 正确
```

类型转换（强制类型转换）：

- **拓宽**（ `int` → `long`、`int` → `double` ）：安全，Java 中自动进行
- **收窄**（ `double` → `int`、`int` → `byte` ）：可能丢失数据，只有显式要求时才进行
 `(int) 3.99 == 3`（截断） · `(byte) 300 == 44`（只保留最低 8 位）
- Go 中**每一次**转换都必须显式写出： `float64(a) / float64(b)`

字符：ASCII 与 Unicode

- **ASCII**: 7 位, 128 个字符 (英文字母、数字、标点)

'A' = 65、'a' = 97、'0' = 48 → 'a' - 'A' == 32

- **Unicode**: 涵盖世界上所有文字, 超过 150 000 个字符

'中' = U+4E2D = 20013 · 'é' = U+00E9 = 233

- **UTF-8**: 变长编码 (1-4 字节) – Web 和 Go 的默认编码

Java

```
char c = 'A';  
System.out.println((int) c);    // 65  
char d = (char) (c + 1);        // 'B'
```

Go

```
c := 'A'           // rune  
fmt.Println(c)      // 65  
fmt.Printf("%c", '中') // 中
```

字符串将在**第 4 周**详细讲解。

布尔类型与逻辑运算

a	b	a and b	a or b	not a
假	假	假	假	真
假	真	假	真	真
真	假	假	真	假
真	真	真	真	假

运算	Java	Go
与	&&	&&
或		
非	!	!
异或	^	!=

- **短路求值：**在 `a && b` 中，如果 `a` 为假，`b` 根本不会被计算
`if (i < n && t[i] > 0)` – 永远不会越过数组末尾！

字面量、变量、常量

- **字面量**：直接写在程序文本中的值

42 、 0x2A 、 0b101010 、 3.14 、 2.5e3 、 'A' 、 "hello" 、 true

- **变量**：有名字的存储位置，程序运行时其值可以改变

- **常量**：不能改变的有名字的值

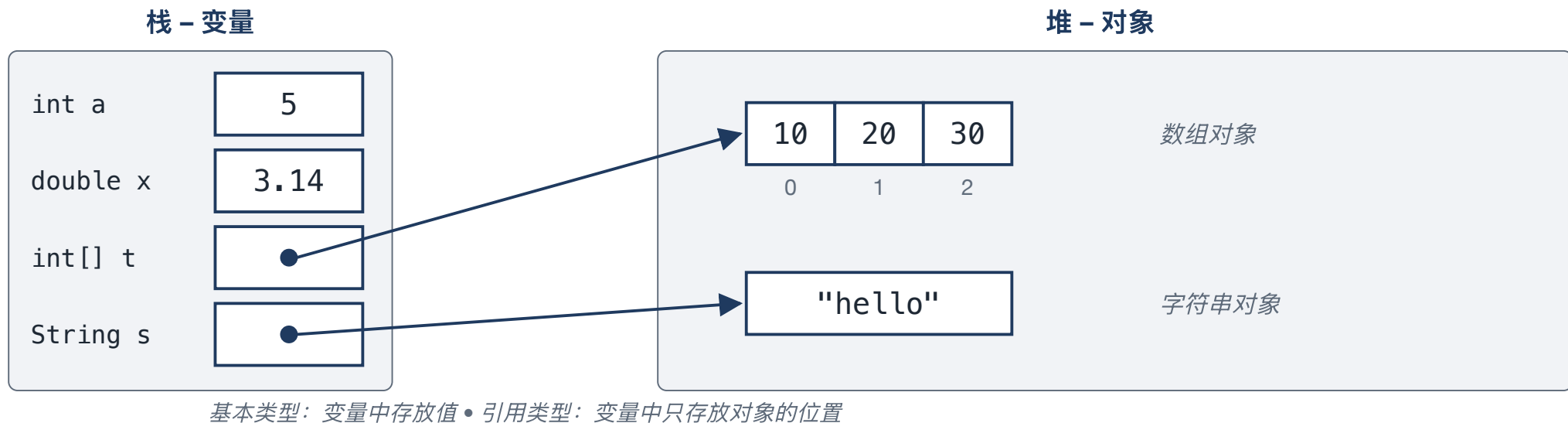
Java

```
int count = 0;           // 变量
final int DAYS = 7;      // 常量
long big = 9_000_000_000L; // L:long
float f = 2.5f;          // f:float
```

Go

```
var count int           // 零值为 0
n := 10                  // 短变量声明
const Days = 7          // 常量
var f float32 = 2.5
```

指针与引用



- **指针：** 值为**内存地址**的数据
- 通过指针可以间接访问另一个数据
- 特殊值：**null / nil** – “不指向任何地方”

Java 中的引用，Go 中的指针

Java

- 基本类型（`int`、`double` ...）：变量中存放**值本身**
- 对象（数组、`String`、自定义类）：变量中存放**引用**
- 没有指针运算，地址不可见

```
int[] a = {10, 20, 30};
int[] b = a;      // 同一个数组!
b[0] = 99;
System.out.println(a[0]); // 99
```

Go

- 所有类型都按值复制 – 甚至数组（`[3]int`）也是
- 指针：`&x` 是地址，`*p` 是所指向的值
- 切片（`[]int`）和 `map` 内部包含引用

```
x := 5
p := &x      // p 中存放地址
*p = 6
fmt.Println(x) // 6
```

指针/引用使**动态数据结构**（链表、树）成为可能 – 见第 9 周起的内容。



数据结构

什么是数据结构？

数据结构是一组逻辑上相关的数据元素，由以下三方面确定：

1. 它的**元素**（及其类型），
2. 元素之间的**关系**，
3. 可以进行的**操作**。

示例 – 班级名单：元素：姓名（字符串） · 关系：按字母顺序（每个人都有前驱和后继） · 操作：查找、插入新名字、删除名字

数据结构可以按多种标准分类 ►

分类：元素的类型与个数

按元素的类型

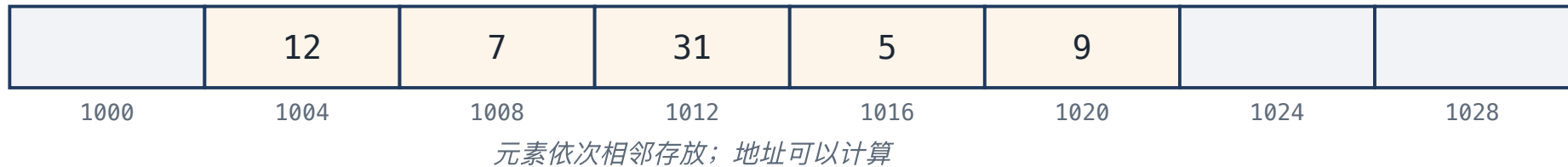
- **同构**：所有元素类型相同 – 数组、字符串、集合
- **异构**：元素可以有不同的类型 – 记录（`class` / `struct`）

按元素的个数

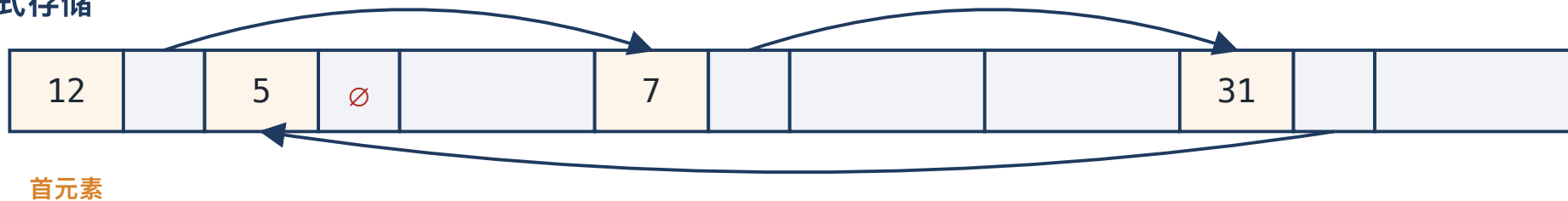
- **静态**：元素个数在创建时确定，以后不再改变
Java 数组 `new int[10]`、Go 数组 `[10]int`
- **动态**：元素个数在运行时可以改变
链表、`ArrayList`、Go 的 `slice`、树

分类：存储方式

顺序存储



链式存储

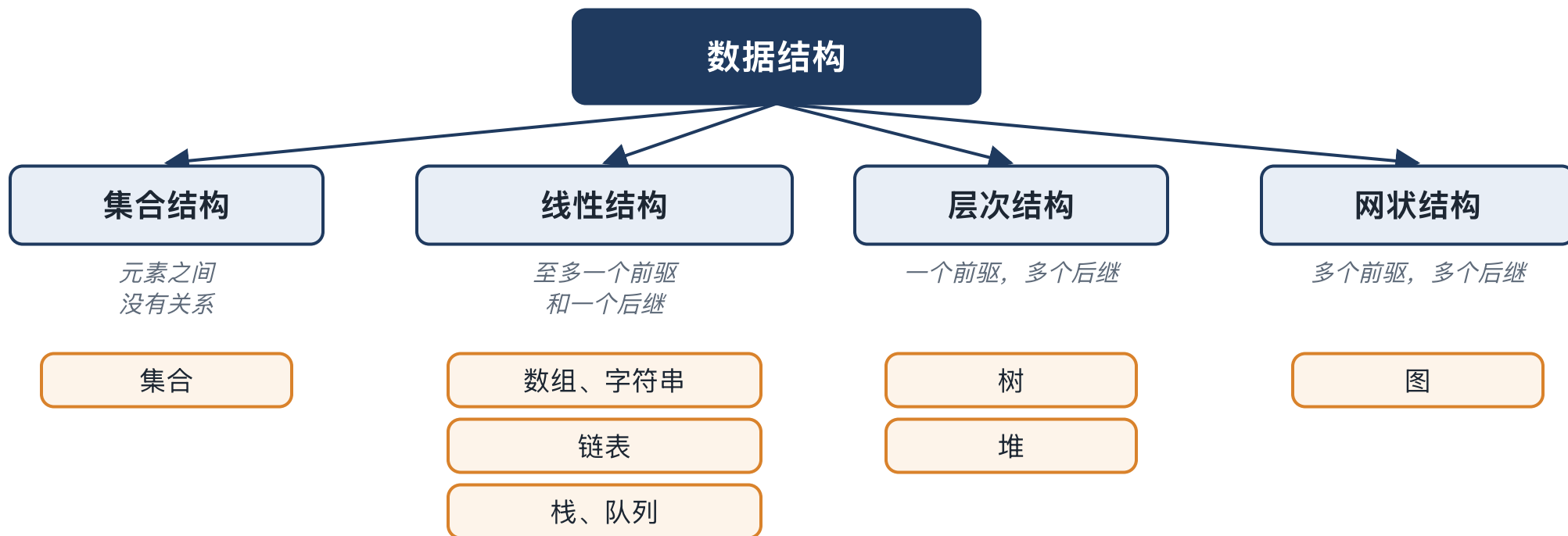


- 顺序存储：第 i 个元素的地址可以计算：基地址 + $(i - 1) \cdot \text{元素大小}$
- 链式存储：每个元素存放下一个元素的位置（指针） - 插入时无需移动元素

分类：元素的访问方式

- **直接（随机）访问：**按下标，访问每个元素的时间相同
数组： `t[1000]` 和 `t[0]` 一样快
- **顺序访问：**只能经过前面的元素才能到达某个元素
链表、（磁带）
- **关联（按内容）访问：**通过键访问元素
哈希表： `map["Wuhan"]`
- **受限访问：**只能访问特定的元素
栈（只能访问栈顶）、队列（只能访问队头）

分类：元素之间的关系



总览：本课程的数据结构

数据结构	元素	大小	存储	访问	关系	周
数组	同构	静态	顺序	直接	线性	3
字符串	同构	静态*	顺序	直接	线性	4
集合	同构	可变	多种	—	无	6
栈、队列	同构	动态	顺序或链式	受限	线性	7、8、11
链表	同构	动态	链式	顺序	线性	9、10
哈希表	同构	动态	混合	关联	无	11
二叉树	同构	动态	链式（或数组）	顺序	层次	14、15
记录	异构	静态	顺序	直接（字段名）	—	—

* Java 和 Go 中的字符串是不可变的。

数据结构上的操作

操作	含义
创建	建立一个空的结构
插入	加入新元素
删除	移除元素
修改	改变某个元素的值
查找	找到具有给定性质的元素
遍历（处理）	每个元素访问一次
排序	把元素排成一定的顺序

如果一个数据结构的最常用操作很快，它就适合这个任务。

抽象数据类型 (ADT)

- **ADT**: 由操作定义的数据结构 – 说明它能**做什么**, 而不是**怎么做**
- 实现 (表示) 可以隐藏, 也可以替换

Java – 接口

```
interface Stack {  
    void push(int x);  
    int pop();  
    boolean isEmpty();  
}
```

Go – 接口

```
type Stack interface {  
    Push(x int)  
    Pop() int  
    IsEmpty() bool  
}
```

同一个栈既可以用**数组**实现 (第 7 周), 也可以用**链表**实现 (第 11 周) – 使用者的代码不需要改变。

小结

- 程序 = 算法 + 数据结构
- 算法是有限的、定义明确的步骤序列；我们用**伪代码**描述它（下标从 1 开始）
- **数据类型** = 值的集合 + 运算 + 表示
- 表示是有局限的：**溢出**、**浮点误差**、**整数除法**
- **指针/引用**存放内存地址
- 数据结构的分类：同构/异构 · 静态/动态 · 顺序/链式 · 直接/顺序访问 · 线性/层次/网状

下周：如何衡量算法的效率？顺序查找与二分查找。

复习题

判断对错：

1. 数组是异构数据结构。
2. 链表的元素按顺序访问。
3. 在 Java 中，`Integer.MAX_VALUE + 1` 会引起运行时错误。
4. 在 Java 中，表达式 `0.1 + 0.2 == 0.3` 的值为假。
5. 指针的值是一个内存地址。
6. 静态数据结构中元素的值不能改变。



实践课

搭建环境

Java

```
java -version          # 17 或更新版本
cd code/java/w01
java Types.java        # 编译 + 运行
```

或者用传统方式：

```
javac Types.java
java Types
```

Go

```
go version             # 1.23 或更新版本
cd code/go
go run ./w01/types
```

格式化与检查：

```
gofmt -w .
go vet ./...
```

示例程序

程序	Java	Go	演示内容
基本类型	<code>Types.java</code>	<code>w01/types</code>	取值范围、字面量、字符编码、常量
常见陷阱	<code>Pitfalls.java</code>	<code>w01/pitfalls</code>	溢出、整数除法、 $0.1 + 0.2$ 、类型转换
值与引用	<code>ValueRef.java</code>	<code>w01/valueref</code>	参数传递、别名、 <code>null/nil</code> 、交换
欧几里得算法	<code>Gcd.java</code>	<code>w01/gcd</code>	伪代码 → 代码，打印每一步

逐个运行它们，并且在运行**之前**先猜一猜会输出什么！我们会讨论出人意料的那几行。

练习

1. **位数**: 编写函数, 返回存放整数 $n \geq 0$ 需要多少位! (`255` $\rightarrow$ `8` , `256` $\rightarrow$ `9`)
2. **二进制形式**: 用反复除以 2 的方法输出一个正整数的二进制形式! (`50` $\rightarrow$ `110010`)
3. **大写字母**: 利用字符编码把小写字母转换成大写字母 (不使用库函数) !
4. **实数比较**: 编写带容差的 `almostEqual(x, y)` 函数!
5. **真值表**: 输出 `and` 、 `or` 、 `not` 和 异或 的真值表!
6. **溢出检查**: 在**相加之前**判断 `x + y` 是否会溢出 (`int` / `int32`) !

参考答案: `Exercises01.java` 、 `w01/exercises`

课后作业

1. Java 中 `(byte) 200` 的值是多少？Go 中 `int8(200)` 呢？先手工计算，再上机验证！
2. 16 位无符号整数可以存放多少个不同的值？最大值是多少？
3. 编写程序，输出文本 `"Hello, 世界"` 的每个字符及其 Unicode 编码！
4. 找出生活中**栈**、**队列**和**树**结构的例子！