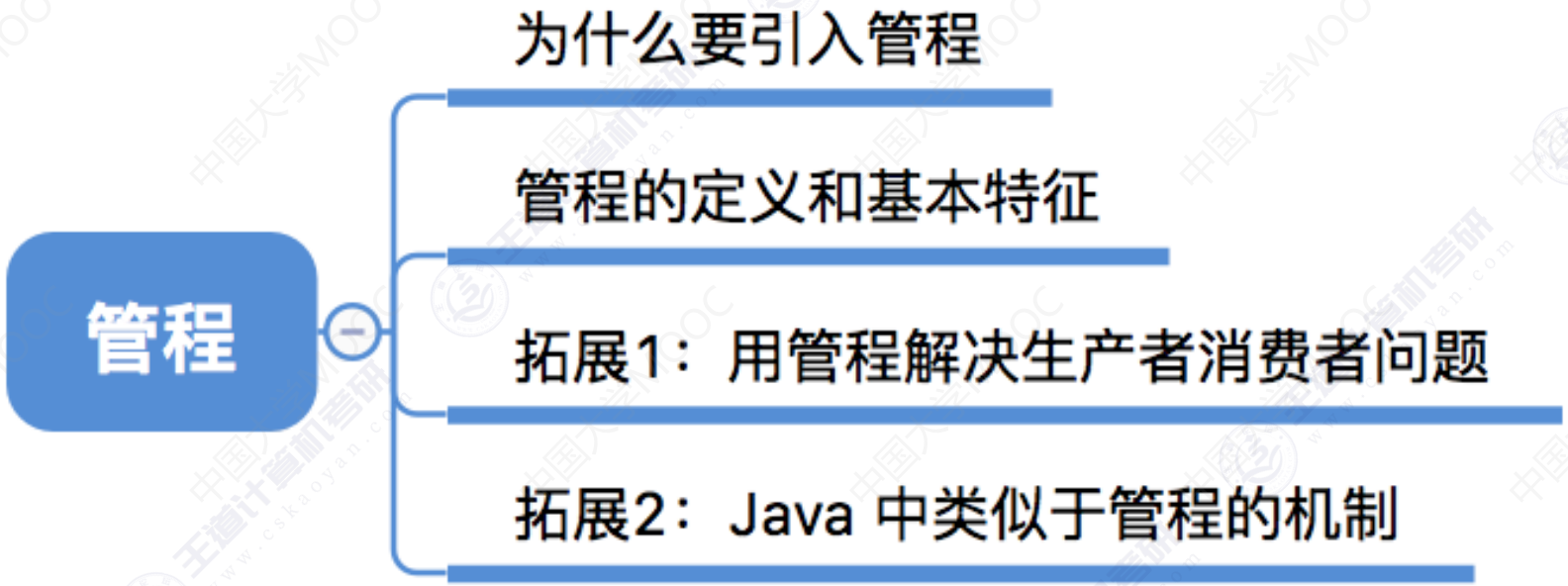
学习建议

建议先学习“2.3.6 经典同步问题”，再回来学习“2.3.5 管程”，更便于理解

本节内容

管程

知识总览



为什么要引入管程

信号量机制存在的问题：编写程序困难、易出错

能不能设计一种机制，让程序员写程序时不需要再关注复杂的PV操作，让写代码更轻松呢？

1973年，Brinch Hansen 首次在程序设计语言 (Pascal) 中引入了“管程”成分——一种高级同步机制

```
producer (){
    while(1){
        生产一个产品;
        P(mutex);        ①
        P(empty);        ②
        把产品放入缓冲区;
        V(mutex);
        V(full);
    }
}
```

```
consumer (){
    while(1){
        P(mutex);        ③
        P(full);         ④
        从缓冲区取出一个产品;
        V(mutex);
        V(empty);
        使用产品;
    }
}
```

像这样如果写错了P操作的顺序，按①②③执行，就会发生死锁

管程的定义和基本特征

管程是一种特殊的软件模块，有这些部分组成：

1. 局部于管程的共享数据结构说明；
2. 对该数据结构进行操作的一组过程；
3. 对局部于管程的共享数据设置初始值的语句；
4. 管程有一个名字。

跨考Tips：“过程”其实就是“函数”

管程的基本特征：

1. 局部于管程的数据只能被局部于管程的过程所访问；
2. 一个进程只有通过调用管程内的过程才能进入管程访问共享数据；
3. 每次仅允许一个进程在管程内执行某个内部过程。

拓展1：用管程解决生产者消费者问题

管程中设置条件变量和等待/唤醒操作，以解决同步问题

```
monitor ProducerConsumer
  condition full, empty;  //条件变量,用于同步(排队)
  int count=0;  //缓冲区中的产品数
  void insert (Item item) { //把产品item放入缓冲区
    if (count == N)
      wait (full);
    count++;
    insert_item (item);
    if (count == 1)
      signal(empty);
  }
  Item remove () {  //从缓冲区中取出一个产品
    if (count == 0)
      wait (empty);
    count--;
    if (count == N-1)
      signal(full);
    return remove_item();
  }
end monitor;
```

由编译器负责实现各进程互斥地进入管程中的过程

insert → producer2

```
//生产者进程
producer (){
  while(1){
    item = 生产一个产品;
    ProdecerConsumer.insert (item);
  }
}
```

```
//消费者进程
consumer (){
  while(1){
    item = ProdecerConsumer.remove ();
    消费产品item;
  }
}
```

empty → consumer1 → consumer2

每次仅允许一个进程在管程内执行某个内部过程。

例1：两个生产者进程并发执行，依次调用了 insert 过程...

例2：两个消费者进程先执行，生产者进程后执行...

拓展1：用管程解决生产者消费者问题

引入管程的目的无非就是要更方便地实现进程互斥和同步。

1. 需要在管程中定义共享数据（如生产者消费者问题的缓冲区）
2. 需要在管程中定义用于访问这些共享数据的“入口”——其实就是一些函数（如生产者消费者问题中，可以定义一个函数用于将产品放入缓冲区，再定义一个函数用于从缓冲区取出产品）
3. 只有通过这些特定的“入口”才能访问共享数据
4. 管程中有很多“入口”，但是每次只能开放其中一个“入口”，并且只能让一个进程或线程进入（如生产者消费者问题中，各进程需要互斥地访问共享缓冲区。管程的这种特性即可保证一个时间段内最多只会有一个进程在访问缓冲区。注意：这种***互斥特性是由编译器负责实现***的，程序员不用关心）
5. 可在管程中设置条件变量及等待/唤醒操作以解决同步问题。可以让一个进程或线程在条件变量上等待（此时，该进程应先释放管程的使用权，也就是让出“入口”）；可以通过唤醒操作将等待在条件变量上的进程或线程唤醒。

程序员可以用某种特殊的语法定义一个管程（比如: monitor ProducerConsumer …… end monitor;），之后其他程序员就可以使用这个管程提供的特定“入口”很方便地使用实现进程同步/互斥了。

“封装”思想

拓展2：Java 中类似于管程的机制

Java 中，如果用关键字 synchronized 来描述一个函数，那么这个函数同一时间段内只能被一个线程调用

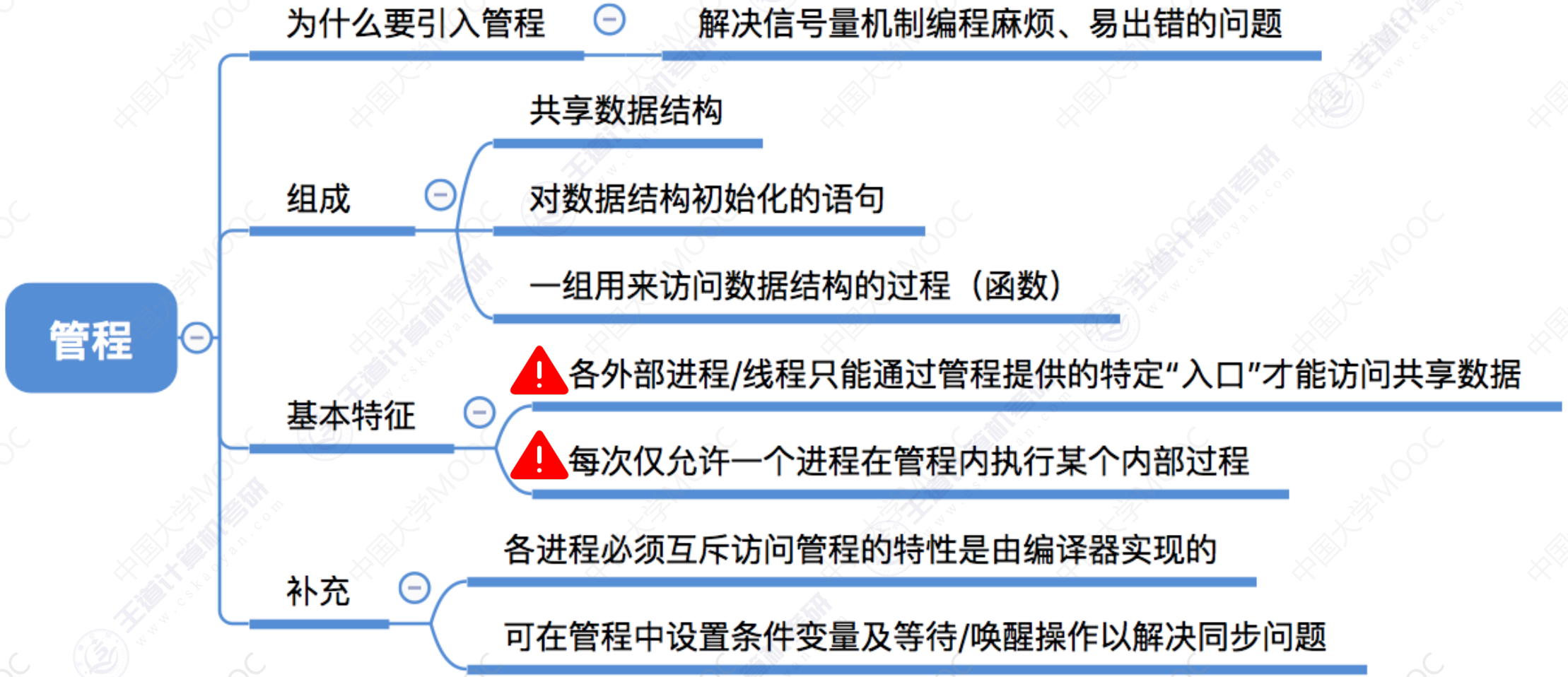
```
static class monitor {
  private Item buffer[] = new Item[N];
  private int count = 0;

  public synchronized void insert (Item item) {
      ......
  }
}
```

每次只能有一个线程进入 insert 函数，如果多个线程同时调用 insert 函数，则后来者需要排队等待

Tips：不熟悉 Java 的同学看不懂也没关系，不会考，仅作为思维拓展。
熟悉 Java 的同学在时间充裕的情况下可以动手尝试用 synchronized 实现生产者消费者问题的“管程”

知识回顾与重要考点



公众号：王道在线

b站：王道计算机教育

抖音：王道计算机考研