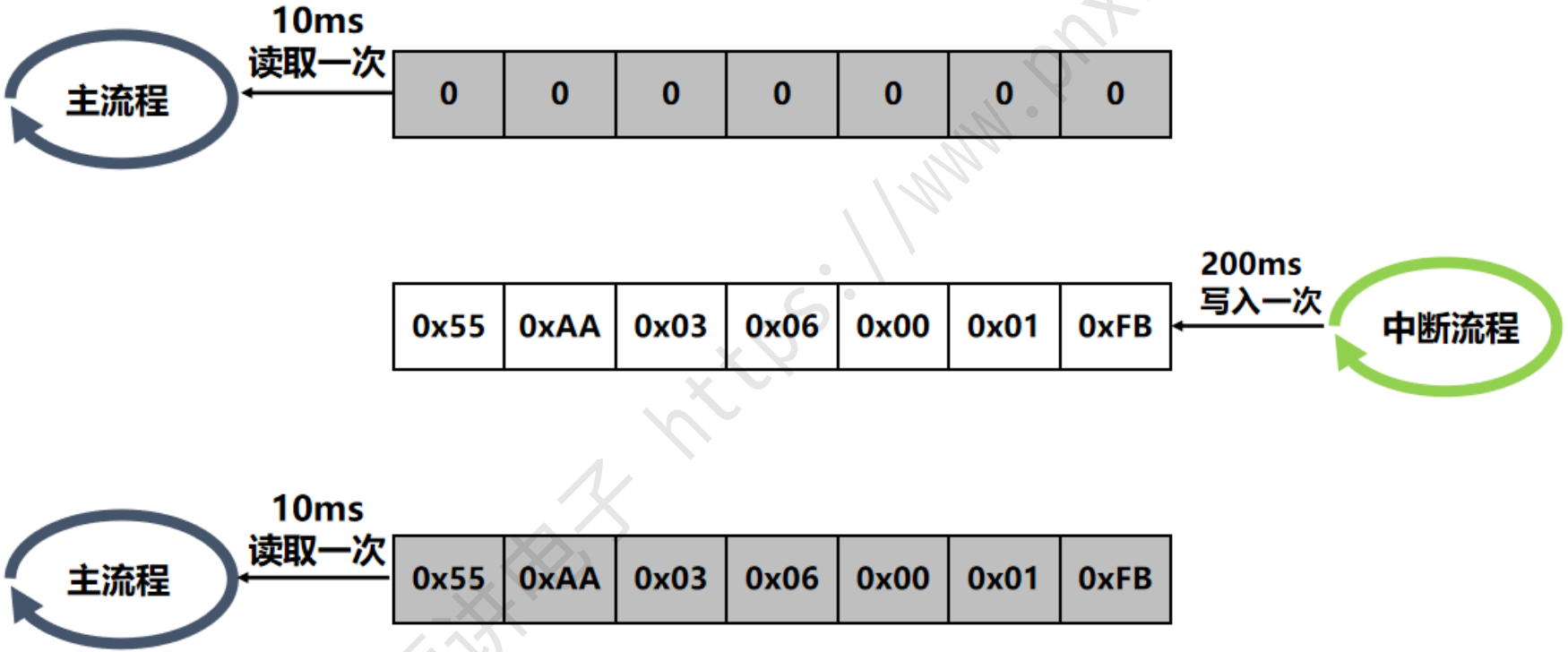
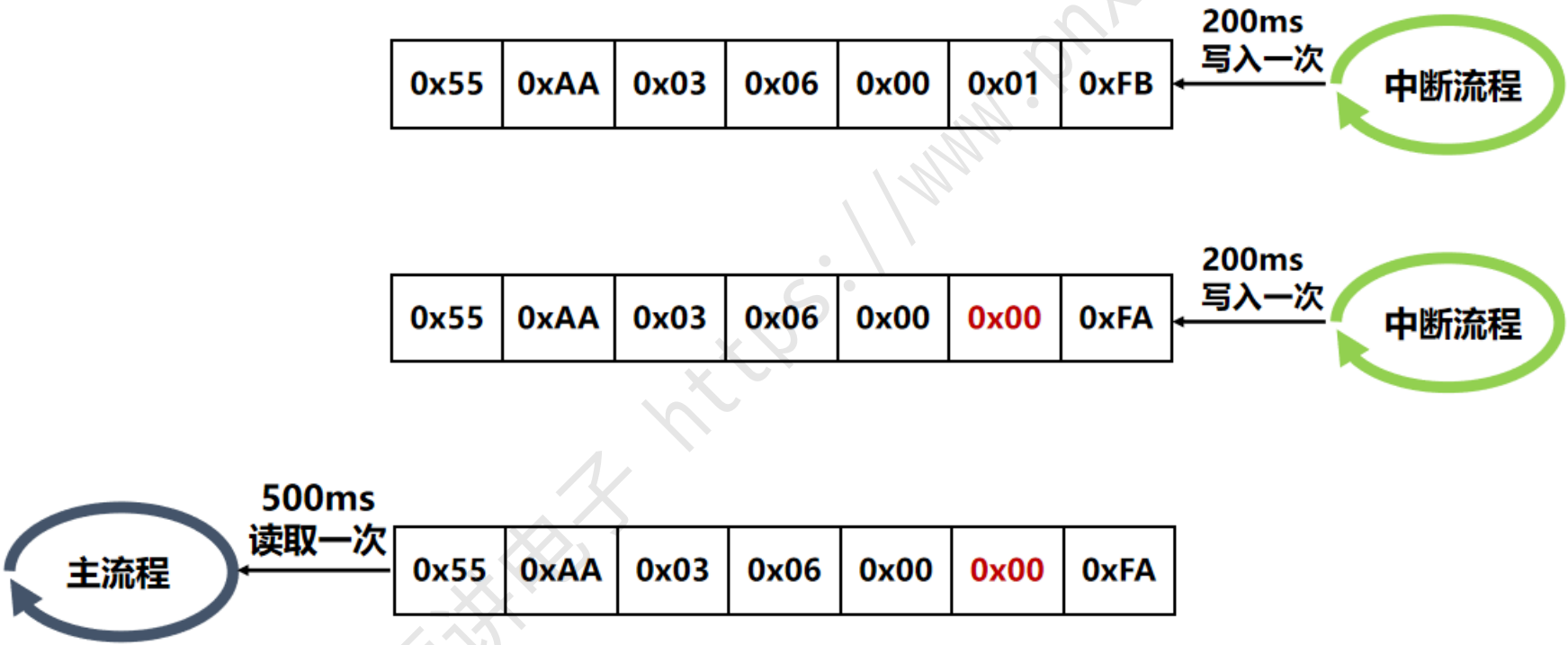
为什么要使用环形队列

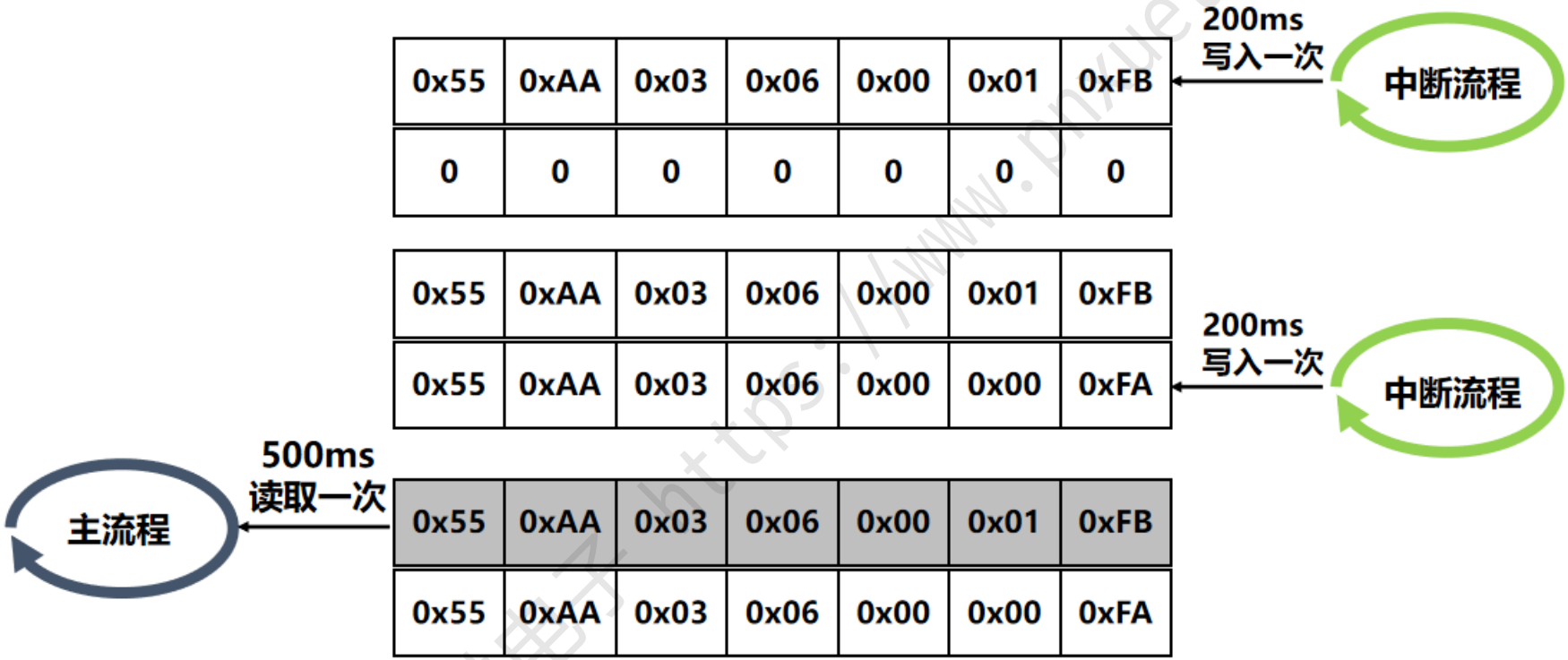
接收到的数据频率太快怎么办?



接收到的数据频率太快怎么办?

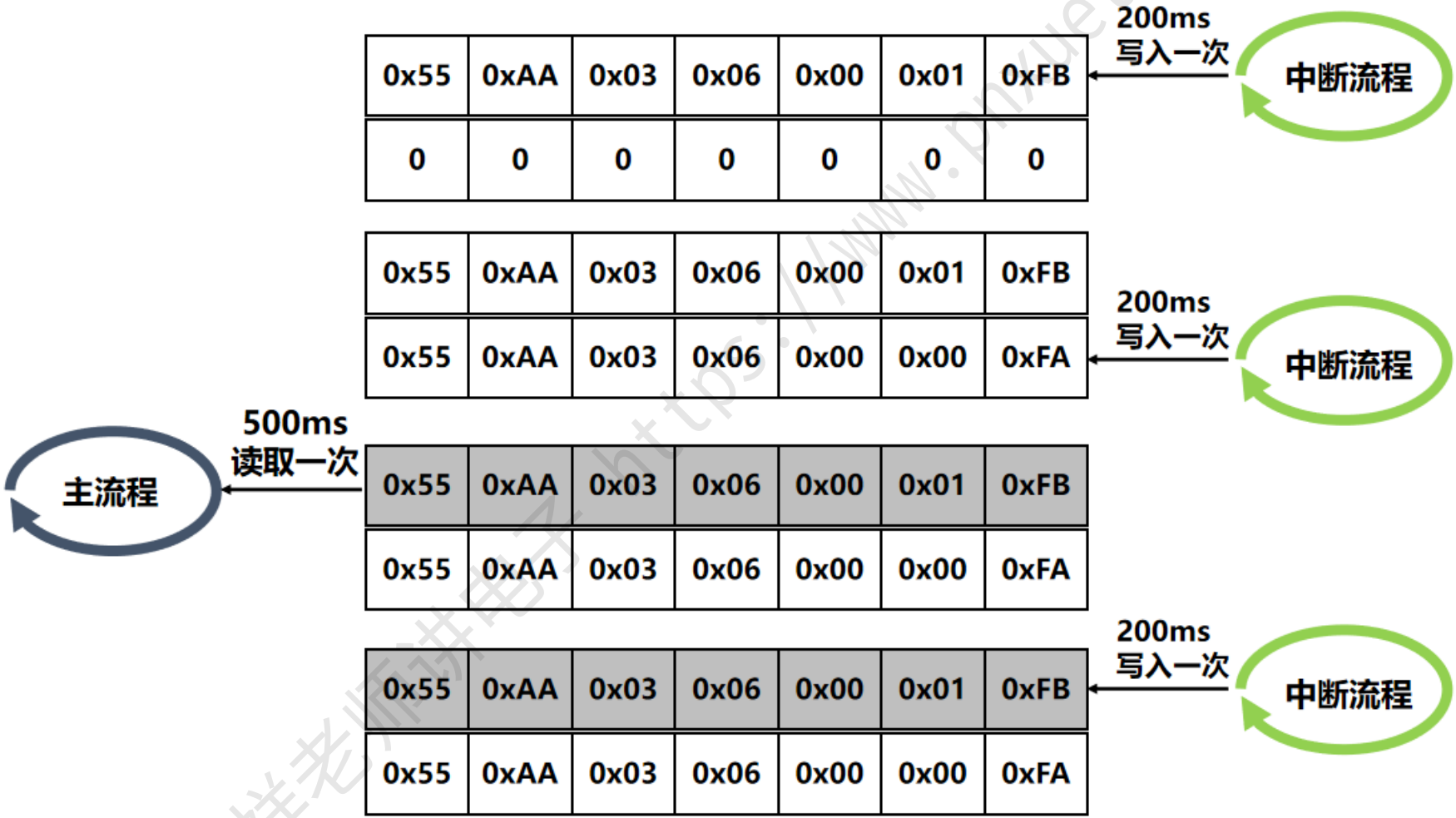


接收到的数据频率太快怎么办?



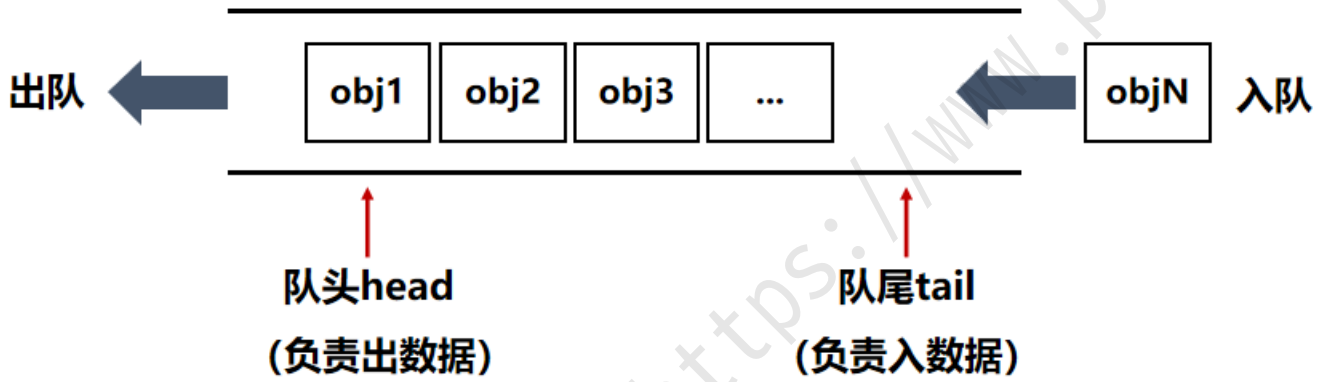
接收到的数据频率太快怎么办?

接收到的数据频率太快怎么办?



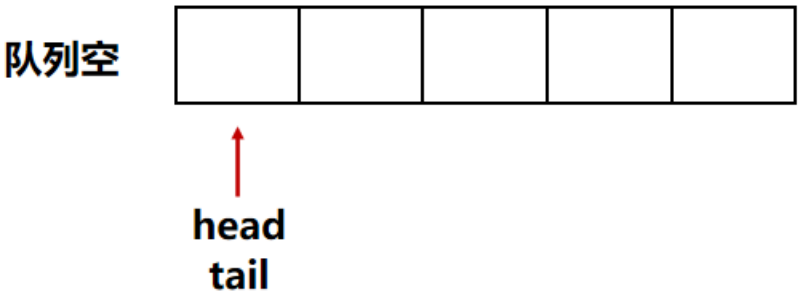
环形队列（循环队列）

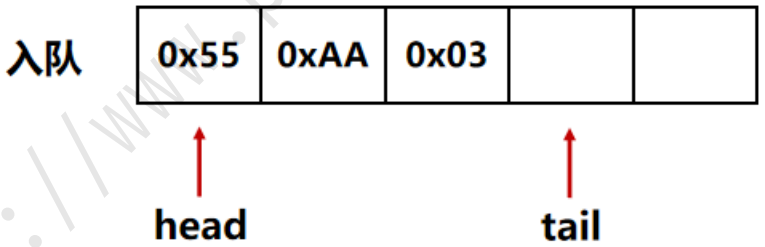
■ 队列：支持数据先进先出，后进后出的一种数据结构，既可以使用数组实现也可以使用链表来实现：

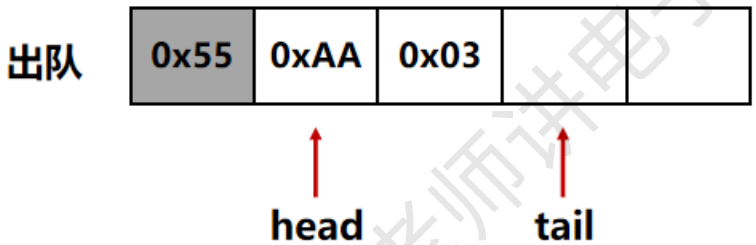


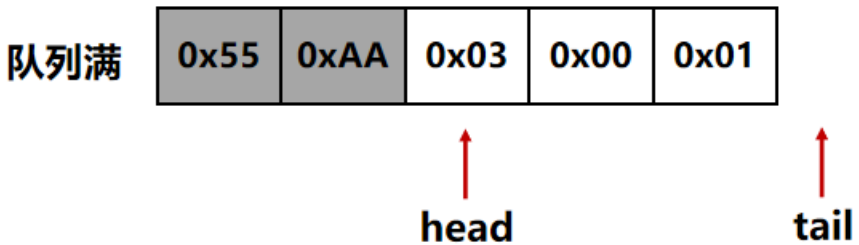
队列基本概念

- head指向队头元素；tail指向下一个入队元素的存储位置，对应数组的下标。



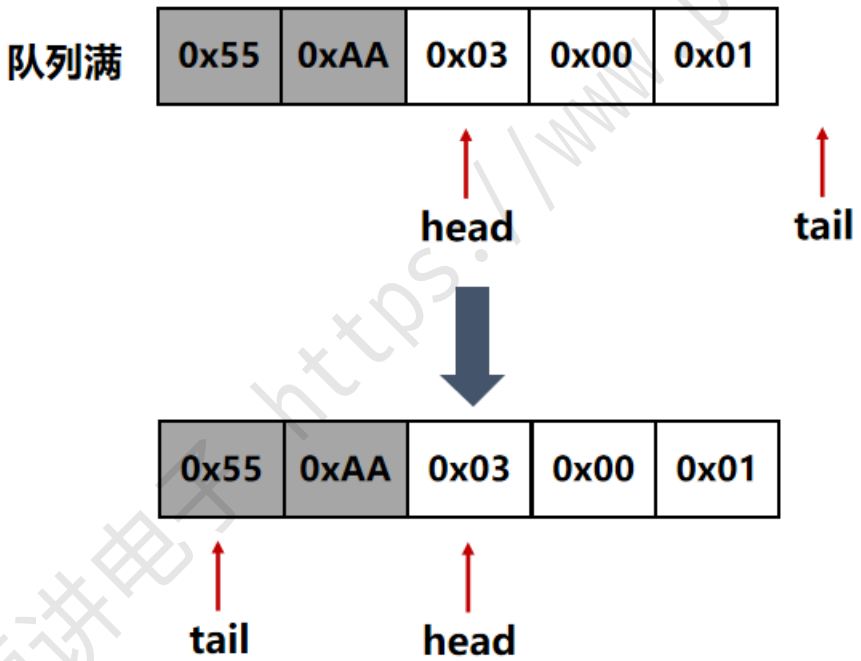






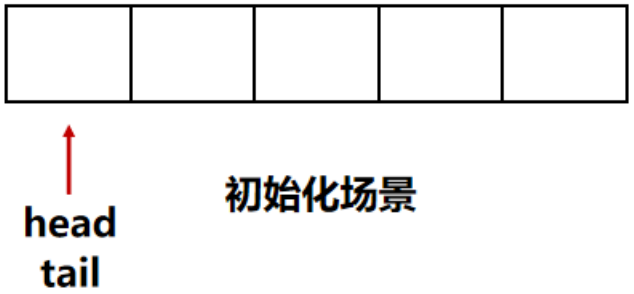
环形队列（循环队列）

■ 环形队列：使队列空间能重复使用

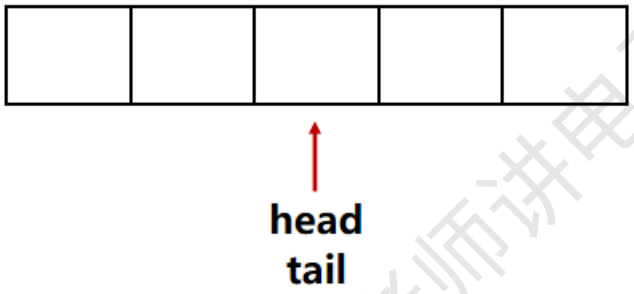


环形队列（循环队列）

- 队列空（head == tail）：

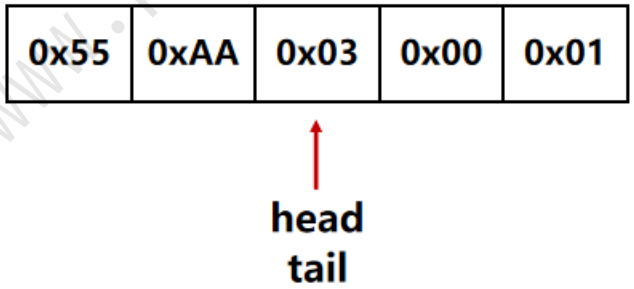


初始化场景



读比写快的场景，head追赶上tail

- 队列满（?）：



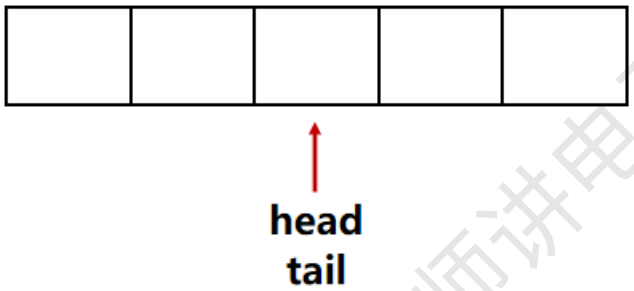
写比读快的场景，tail追赶上head

➢ 如何区别两个判断条件重合问题?

环形队列（循环队列）

- 队列空（head == tail）：

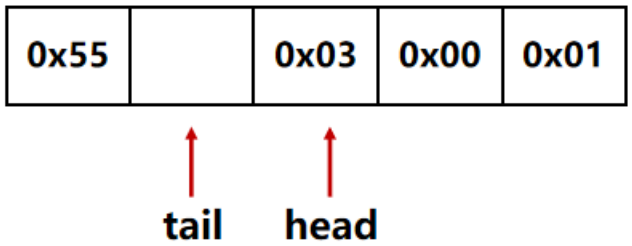
初始化场景



读比写快的场景，head追赶上tail

✓ 为了区别两个判断条件重合问题，可以设定当tail指向head前一个位置时为满，这样会在数组中留一个空的位置，不写数据

- 队列满（当tail指向head前一个位置时）：



写比读快的场景，tail追赶上head

环形队列（循环队列）

- **数据结构：**

```
typedef struct
{
    uint32_t head;      //数组下标，指向队头
    uint32_t tail;      //队列尾下标，指向队尾
    uint32_t size;      //队列缓存长度（初始化时赋值）
    uint8_t *buffer;    //队列缓存数组（初始化时赋值）
} QueueType_t;
```

```
/**
*****************************************************************
* @brief  初始化（创建）队列，每个队列须先执行该函数才能使用
* @param  queue, 队列变量指针
* @param  buffer, 队列缓存区地址
* @param  size, 队列缓存区长度
* @return
*****************************************************************
*/
void QueueInit(QueueType_t *queue, uint8_t *buffer, uint32_t size)
{
    queue->buffer = buffer;
    queue->size = size;
    queue->head = 0;
    queue->tail = 0;
}
```

```
#define MAX_BUF_SIZE 77
static volatile uint8_t g_rcvDataBuf[MAX_BUF_SIZE];
static QueueType_t g_rcvQueue;
```

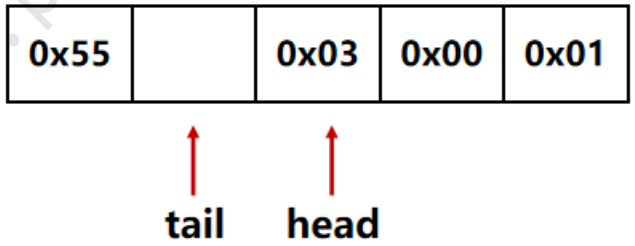
```
QueueInit(&g_rcvQueue, g_rcvDataBuf, MAX_BUF_SIZE);
```

环形队列（循环队列）

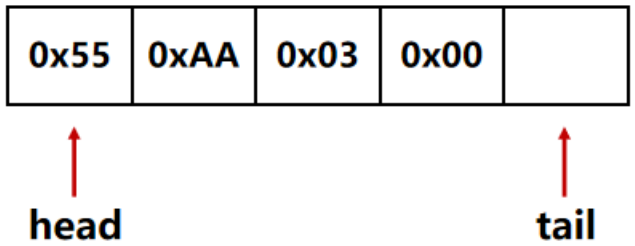
```
/**
*****************************************************************
* @brief  压入数据到队列中
* @param  queue, 队列变量指针
* @param  data, 待压入队列的数据
* @return 压入队列是否成功
*****************************************************************
*/
QueueStatus_t QueuePush(QueueType_t *queue, uint8_t data)
{
   uint32_t index = (queue->tail + 1) % queue->size;

   if (index == queue->head)
   {
      return QUEUE_OVERLOAD;
   }
   queue->buffer[queue->tail] = data;
   queue->tail = index;
   return QUEUE_OK;
}
```

- **队列满（当tail指向head前一个位置时）：**



➢ **tail + 1 == head**



➢ **(tail + 1) % size == head**

环形队列（循环队列）

```
/**
*****************************************************************
* @brief   从队列中弹出数据
* @param   queue, 队列变量指针
* @param   pdata, 待弹出队列的数据缓存地址
* @return  弹出队列是否成功
*****************************************************************
*/
QueueStatus_t QueuePop(QueueType_t *queue, uint8_t *pdata)
{
    if(queue->head == queue->tail)
    {
        return QUEUE_EMPTY;
    }

    *pdata = queue->buffer[queue->head];
    queue->head = (queue->head + 1) % queue->size;
    return QUEUE_OK;
}
```

环形队列（循环队列）

```
/**
*****************************************************************
* @brief   压入一组数据到队列中
* @param   queue, 队列变量指针
* @param   pArray, 待压入队列的数组地址
* @param   len, 待压入队列的元素个数
* @return  实际压入到队列的元素个数
*****************************************************************
*/
uint32_t QueuePushArray(QueueType_t *queue, uint8_t *pArray, uint32_t len)
{
    uint32_t i;
    for (i = 0; i < len; i++)
    {
        if(QueuePush(queue, pArray[i]) == QUEUE_OVERLOAD)
        {
            break;
        }
    }
    return i;
}
```

环形队列（循环队列）

```
/**
*****************************************************************
* @brief   从队列中弹出一组数据
* @param   queue, 队列变量指针
* @param   pArray, 待弹出队列的数据缓存地址
* @param   len, 待弹出队列的数据的最大长度
* @return  实际弹出数据的数量
*****************************************************************
*/
uint32_t QueuePopArray(QueueType_t *queue, uint8_t *pArray, uint32_t len)
{
    uint32_t i;
    for(i = 0; i < len; i++)
    {
        if (QueuePop(queue, &pArray[i]) == QUEUE_EMPTY)
        {
            break;
        }
    }
    return i;
}
```

THANK YOU!