using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

/// <summary>

/// 字节缓冲类

/// </summary>

public class ByteBuffer

{

//字节缓存区

private byte[] buf;

//读取索引

private int readIndex = 0;

//写入索引

private int writeIndex = 0;

//读取索引标记

private int markReadIndex = 0;

//写入索引标记

private int markWirteIndex = 0;

//缓存区字节数组的长度

private int capacity;

/// <summary>

/// 构造方法

/// </summary>

/// <param name="capacity">缓存字节数组的长度</param>

public ByteBuffer(int capacity)

{

buf = new byte[capacity];

this.capacity = capacity;

}

/// <summary>

/// 构造方法

/// </summary>

/// <param name="bytes">字节数组</param>

public ByteBuffer(byte[] bytes)

{

buf = bytes;

this.capacity = bytes.Length;

}

/// <summary>

/// 构建一个capacity长度的字节缓存区ByteBuffer对象

/// </summary>

/// <param name="capacity">缓存字节数组的长度</param>

/// <returns>ByteBuffer实例</returns>

public static ByteBuffer Allocate(int capacity)

{

return new ByteBuffer(capacity);

}

/// <summary>

/// 构建一个以bytes为字节缓存区的ByteBuffer对象，一般不推荐使用

/// </summary>

/// <param name="bytes">字节数组</param>

/// <returns>ByteBuffer实例</returns>

public static ByteBuffer Allocate(byte[] bytes)

{

return new ByteBuffer(bytes);

}

/// <summary>

/// 根据length长度，确定大于此leng的最近的2次方数，如length=7，则返回值为8

/// </summary>

/// <param name="length">长度</param>

/// <returns>int</returns>

private int FixLength(int length)

{

int n = 2;

int b = 2;

while (b < length)

{

b = 2 << n;

n++;

}

return b;

}

/// <summary>

/// 翻转字节数组，如果本地字节序列为低字节序列，则进行翻转以转换为高字节序列

/// </summary>

/// <param name="bytes">字节数组</param>

/// <returns>byte[]</returns>

private byte[] flip(byte[] bytes)

{

if (BitConverter.IsLittleEndian)

{

Array.Reverse(bytes);

}

return bytes;

}

/// <summary>

/// 确定内部字节缓存数组的大小

/// </summary>

/// <param name="currLen"></param>

/// <param name="futureLen"></param>

/// <returns>int</returns>

private int FixSizeAndReset(int currLen, int futureLen)

{

if (futureLen > currLen)

{

//以原大小的2次方数的两倍确定内部字节缓存区大小

int size = FixLength(currLen) \* 2;

if (futureLen > size)

{

//以将来的大小的2次方的两倍确定内部字节缓存区大小

size = FixLength(futureLen) \* 2;

}

byte[] newbuf = new byte[size];

Array.Copy(buf, 0, newbuf, 0, currLen);

buf = newbuf;

capacity = newbuf.Length;

}

return futureLen;

}

/// <summary>

/// 将bytes字节数组从startIndex开始的length字节写入到此缓存区

/// </summary>

/// <param name="bytes"></param>

/// <param name="startIndex"></param>

/// <param name="length"></param>

public void WriteBytes(byte[] bytes, int startIndex, int length)

{

lock (this)

{

int offset = length - startIndex;

if (offset <= 0) return;

int total = offset + writeIndex;

int len = buf.Length;

FixSizeAndReset(len, total);

for (int i = writeIndex, j = startIndex; i < total; i++, j++)

{

buf[i] = bytes[j];

}

writeIndex = total;

}

}

/// <summary>

/// 将字节数组中从0到length的元素写入缓存区

/// </summary>

/// <param name="bytes"></param>

/// <param name="length"></param>

public void WriteBytes(byte[] bytes, int length)

{

WriteBytes(bytes, 0, length);

}

/// <summary>

/// 将字节数组全部写入缓存区

/// </summary>

/// <param name="bytes"></param>

public void WriteBytes(byte[] bytes)

{

WriteBytes(bytes, bytes.Length);

}

/// <summary>

/// 将一个ByteBuffer的有效字节区写入此缓存区中

/// </summary>

/// <param name="buffer"></param>

public void Write(ByteBuffer buffer)

{

if (buffer == null) return;

if (buffer.ReadableBytes() <= 0) return;

WriteBytes(buffer.ToArray());

}

/// <summary>

/// 将一个字符串写入此缓存区中

/// </summary>

/// <param name="value"></param>

public void WirteUTFString(string value) {

WriteBytes(Encoding.UTF8.GetBytes(value));

}

/// <summary>

/// 写入一个int16数据

/// </summary>

/// <param name="value"></param>

public void WriteShort(short value)

{

WriteBytes(flip(BitConverter.GetBytes(value)));

}

/// <summary>

/// 写入一个uint16数据

/// </summary>

/// <param name="value"></param>

public void WriteUshort(ushort value)

{

WriteBytes(flip(BitConverter.GetBytes(value)));

}

/// <summary>

/// 写入一个int32数据

/// </summary>

/// <param name="value"></param>

public void WriteInt(int value)

{

//byte[] array = new byte[4];

//for (int i = 3; i >= 0; i--)

//{

// array[i] = (byte)(value & 0xff);

// value = value >> 8;

//}

//Array.Reverse(array);

//Write(array);

WriteBytes(flip(BitConverter.GetBytes(value)));

}

/// <summary>

/// 写入一个uint32数据

/// </summary>

/// <param name="value"></param>

public void WriteUint(uint value)

{

WriteBytes(flip(BitConverter.GetBytes(value)));

}

/// <summary>

/// 写入一个int64数据

/// </summary>

/// <param name="value"></param>

public void WriteLong(long value)

{

WriteBytes(flip(BitConverter.GetBytes(value)));

}

/// <summary>

/// 写入一个uint64数据

/// </summary>

/// <param name="value"></param>

public void WriteUlong(ulong value)

{

WriteBytes(flip(BitConverter.GetBytes(value)));

}

/// <summary>

/// 写入一个float数据

/// </summary>

/// <param name="value"></param>

public void WriteFloat(float value)

{

WriteBytes(flip(BitConverter.GetBytes(value)));

}

/// <summary>

/// 写入一个byte数据

/// </summary>

/// <param name="value"></param>

public void WriteByte(byte value)

{

lock (this)

{

int afterLen = writeIndex + 1;

int len = buf.Length;

FixSizeAndReset(len, afterLen);

buf[writeIndex] = value;

writeIndex = afterLen;

}

}

/// <summary>

/// 写入一个double类型数据

/// </summary>

/// <param name="value"></param>

public void WriteDouble(double value)

{

WriteBytes(flip(BitConverter.GetBytes(value)));

}

/// <summary>

/// 读取一个字节

/// </summary>

/// <returns>byte</returns>

public byte ReadByte()

{

byte b = buf[readIndex];

readIndex++;

return b;

}

/// <summary>

/// 从读取索引位置开始读取len长度的字节数组

/// </summary>

/// <returns>byte[]</returns>

private byte[] Read(int len)

{

byte[] bytes = new byte[len];

Array.Copy(buf, readIndex, bytes, 0, len);

if (BitConverter.IsLittleEndian)

{

Array.Reverse(bytes);

}

readIndex += len;

return bytes;

}

/// <summary>

/// 读取一个uint16数据

/// </summary>

/// <returns>ushort</returns>

public ushort ReadUshort()

{

return BitConverter.ToUInt16(Read(2), 0);

}

/// <summary>

/// 读取一个int16数据

/// </summary>

/// <returns>short</returns>

public short ReadShort()

{

return BitConverter.ToInt16(Read(2), 0);

}

/// <summary>

/// 读取一个uint32数据

/// </summary>

/// <returns>uint</returns>

public uint ReadUint()

{

return BitConverter.ToUInt32(Read(4), 0);

}

/// <summary>

/// 读取一个int32数据

/// </summary>

/// <returns>int</returns>

public int ReadInt()

{

return BitConverter.ToInt32(Read(4), 0);

}

/// <summary>

/// 读取一个uint64数据

/// </summary>

/// <returns>ulong</returns>

public ulong ReadUlong()

{

return BitConverter.ToUInt64(Read(8), 0);

}

/// <summary>

/// 读取一个long数据

/// </summary>

/// <returns>long</returns>

public long ReadLong()

{

return BitConverter.ToInt64(Read(8), 0);

}

/// <summary>

/// 读取一个float数据

/// </summary>

/// <returns>float</returns>

public float ReadFloat()

{

return BitConverter.ToSingle(Read(4), 0);

}

/// <summary>

/// 读取一个double数据

/// </summary>

/// <returns>double</returns>

public double ReadDouble()

{

return BitConverter.ToDouble(Read(8), 0);

}

/// <summary>

/// 从读取索引位置开始读取len长度的字节到disbytes目标字节数组中

/// </summary>

/// <param name="disstart">目标字节数组的写入索引</param>

/// <param name="len">读取长度</param>

public void ReadBytes(byte[] disbytes, int disstart, int len)

{

int size = disstart + len;

for (int i = disstart; i < size; i++)

{

disbytes[i] = this.ReadByte();

}

}

/// <summary>

/// 清除已读字节并重建缓存区

/// </summary>

public void DiscardReadBytes()

{

if (readIndex <= 0) return;

int len = buf.Length - readIndex;

byte[] newbuf = new byte[len];

Array.Copy(buf, readIndex, newbuf, 0, len);

buf = newbuf;

writeIndex -= readIndex;

markReadIndex -= readIndex;

if (markReadIndex < 0)

{

markReadIndex = readIndex;

}

markWirteIndex -= readIndex;

if (markWirteIndex < 0 || markWirteIndex < readIndex || markWirteIndex < markReadIndex)

{

markWirteIndex = writeIndex;

}

readIndex = 0;

}

/// <summary>

/// 清空此对象

/// </summary>

public void Clear()

{

buf = new byte[buf.Length];

readIndex = 0;

writeIndex = 0;

markReadIndex = 0;

markWirteIndex = 0;

}

/// <summary>

/// 设置开始读取的索引

/// </summary>

/// <param name="index"></param>

public void SetReaderIndex(int index)

{

if (index < 0) return;

readIndex = index;

}

/// <summary>

/// 标记读取的索引位置

/// </summary>

/// <param name="index"></param>

public void MarkReaderIndex()

{

markReadIndex = readIndex;

}

/// <summary>

/// 标记写入的索引位置

/// </summary>

public void MarkWriterIndex()

{

markWirteIndex = writeIndex;

}

/// <summary>

/// 将读取的索引位置重置为标记的读取索引位置

/// </summary>

public void ResetReaderIndex()

{

readIndex = markReadIndex;

}

/// <summary>

/// 将写入的索引位置重置为标记的写入索引位置

/// </summary>

public void ResetWriterIndex()

{

writeIndex = markWirteIndex;

}

/// <summary>

/// 可读的有效字节数

/// </summary>

public int ReadableBytes()

{

return writeIndex - readIndex;

}

/// <summary>

/// 获取可读的字节数组

/// </summary>

public byte[] ToArray()

{

byte[] bytes = new byte[writeIndex];

Array.Copy(buf, 0, bytes, 0, bytes.Length);

return bytes;

}

/// <summary>

/// 获取缓存区大小

/// </summary>

public int GetCapacity()

{

return this.capacity;

}

}