Byte和byte[]数组，“表示一个 8 位无符号整数, 一般为8位二进制数”。
Byte是计算机最基础的存储单位和最基础的通讯单位。
而所有的类型都是支持由byte[]类型转换而来。

为什么说Byte是最基础类型那，
其实这里的关键所在是，计算机最基础的算法是编码，包括媒体文件、图片、各种文档以致计算机算有数据的文件展现都是字符串，而这些字符串都是由编码得来。

不管你是各种千奇百怪的字符串组成的格式，最终都要根据编码表，字符转换成相对应的十进制数字，而这相应的十进制数字那，就会存储在byte[]数组中(这里的十进制数是用来计算的，存储时会自动转换为二进制数字存储，程序计算时会按十进制计算)。

媒体文件、图片、各种文档等等—》源文件由字符串组成，多个字符—》单个字符—》十进制数字(根据相应编码)—》byte[]数组(单个字符) —》byte[]数组(多个字符) —》媒体文件、图片、各种文档等等存储单位或通讯单位。

此时这个“单个字符”二进制的长度就是根据相应编码得来的，由多个字符组成的**byte[]就组成了**媒体文件、图片、各种文档等等的源文件。

具体如下：
UTF-8编码：一个英文字符等于一个字节，一个中文（含繁体）等于三个字节。
Unicode编码：一个英文等于两个字节，一个中文（含繁体）等于**两个**字节。

下面可以看到**UTF-8编码的byte[]数组(C# byte[]默认存储计算为十进制)，所以说最终计算机数据是由byte类型组成数据为基础的。**

网络传输：

如果是自己程序间的对接，byte[]类型理论上就够了（实际上也不够，哈哈），在传输段传输byte[]类型，收取端收取byte[]类型，然后转化为原格式就可以了。

“a张”编码—>byte[0]=97、byte[1]=229、byte[2]=188、byte[3]=160（byte[0]=01100001、byte[1]=11100101、byte[2]=10111100、byte[3]=10100000）—>传输—>收取为byte[]—> 解码“a张”。

但是如果传输的数据是开放式的，或者生成可读的文件，比如文件格式的，url，xml、json的话，是不能存储byte[]型的，看不懂也没法解析，这就要求再吧byte[]再编码为字符串类型。

到了网络传输过程了，前文我们已经得到了文件最基础的存储格式，byte[]数组，但作为程序传输他面临个问题，因为程序传输一般可见的是以字符形式传输的（中间过程是网络层自动转换的），那么就需要把byte[]类型转换为字符串传输，很多人是感觉怎么有转换为字符串了，这不是和原来一样了吗，其实不是，现在编码的就不是任意字符串了，是基础的二进制数字10做成的字符串，相当于把byte[]类型每个byte转换为字符串传送，而在接收端在转换为byte[]数组，这相当于无编码过程。

byte[]是数据类型的，长度是有限制的，也不灵活。所以说基础传输其实是1和0组成的长字符串。
简单的说就是数据编码成byte[]，byte[]再二次编码成为易于传送的字符串。
如下
“a张”编码—>byte[0]=97、byte[1]=229、byte[2]=188、byte[3]=160（byte[0]=01100001、byte[1]=11100101、byte[2]=10111100、byte[3]=10100000）—>不编码
“01100001111001011011110010100000” —>传输—>解析成为byte[]—>解码“a张”

“a张”编码—>byte[0]=97、byte[1]=229、byte[2]=188、byte[3]=160（byte[0]=01100001、byte[1]=11100101、byte[2]=10111100、byte[3]=10100000）—> base64 编码串—>传输—>解析成为byte[]—>解码“a张”

更多的是，在转换字符串传输的过程，还要对这个byte[]进行编码。
注1：在传输过程中，基础的byte[]类型还可以做各种编码传输，如base64字符串、16进制编码、xml、json、html等等，不过万变不离其中的是编码和解码。

注2：“传输”时，还是以基础二进制数据传输，不过这和应用层无关，是传输层自动完成了，应用层的数据只需要处理自己的传输格式就可以了。