- Buffer
  - Buffer结构
    - Buffer是一个像Array的对象，但主要用于字节
    - 模块结构
      - Buffer是一个典型的JavaScript与C++结合的模块，他将性能相关部分用C++实现，将非性能部分用JS实现
        - js核心模块： Buffer/SlowBuffer
        - C++内建模块 node_buffer
    - 对象结构
      - 不同编码的字符串占用的元素个数不同
        - 中文UTF-8下占3个元素，字母和半角标点占一个
      - Buffer对象类似于数组，他的元素为16进制的两位数，几 0-255
      - Buffer受Array影响很大
        - 通过length获得长度
        - 通过下标赋值
        - 通过下标访问元素
    - Buffer内存分配
      - Buffer内存的分配不是在V8中，而是在Node的C++层面实现的内存的申请
      - C++层面上申请内存，在JS中分配内存的策略。
      - Node以8K为分界线，区分Buffer是大对象还是小对象
        - 小对象
          - 为了高效使用申请的内存采用slab分配策略(一块预先申请好的固定大小的内存区域)。
            - 每个slab的大小就是8K，在JS层面，以它作为单位单元进行内存分配。
            - full:完全分配状态
            - partia:部分分配状态
            - empty:没有被分配状态
            - 操作
              - 将新申请的SlowBuffer对象指向它
              - 构建小buffer时检查pool
                - 没有则创建一个新的slab单元指向它，同时当前Buffer对象的parent属性指向该slab,并记录下从这个slab的哪个位置开始使用的，slab对象自身也记录被使用了多少字节
              - 再次创建Buffer，判断这个slab的剩余空间是否足够，足够则使用剩余空间，并更新slab分配状态，
              - 否则创建新的slab,原slab剩余空间则浪费
        - 大对象
          - 直接使用c++层面上的内存，无需细腻操作
            - 直接分配一个SlowBuffer对象作为slab单元，这个slab对象会被这个大buffer对象独占
  - Buffer转换
    - 字符串转Buffer
      - 通过构造函数ew Buffer(str, [encoding]); 默认UTF-8
      - buf.write(string, [offset], [length], [encoding])可以存储多种编码类型
    - Buffer转字符串
      - buf.toString([encoding], [start], [end])
    - 不支持的编码类型
      - isEncoding()判断是否支持编码类型
      - 对于不的编，可以Node生中的icon 和viconv-lite
  - Buffer的拼接
    - 乱码产生原因
      - Data事件中，data += chunk;隐藏了toString()的操作会导致宽字节的中文出现乱码等问题
      - 文件可读流在读取时会逐个读取Buffer
      - 假如我们限定Buffer对象的长度{highWaterMark:11},而中文在UTF-8下占3个字节，则最后2个字节会显示乱码
    - setEncoding()
      - 可以设置编码的方法setEncoding()让data事件产地的是编码后的字符串，but还是存在宽字节被截断的问题，所以乱码解决需要decoder
      - decoder对象来自string_decoder模块StringDecoder的实例对象
      - 原理：假如阶段的某个中文前两个字节会被保存到StringDecoder实例内部，第二次write()时，再将这两个字节和后续的组合在一起
      - setEncoding只能解决UTF-8 Base64 UCS-2/UTF-16LE三种，不能解决最终问题
    - 正确拼接buffer
      - 原理：将多个小Buffer对象拼接成一个大的Buffer对象，然后通过iconv-lite等模块来转码
      - 1. 用一个数组来存储接受到的所有Buffer片段并记录下所有片段总长度
      - 2. 调用Buffer.concat()方法生成一个合并的Buffer对象
  - Buffer与性能
    - 应用中我们一般都是操作字符串，但是网络中传世，都需要转为Buffer以二进制数据传输
    - 提高字符串到Buffer的转换效率可以很大程度提高网络吞吐率
    - 通过预先读取静态资源为Buffer可以有效率的减少CPU的重复使用，节约服务器资源
      - 在node构建的web中，将动态内容和静态内容分离，静态内容预先转换buffer
      - 由于文件自身是二进制数据，在不需改变内容的情景下，尽量只读取Buffer，然后传输，不做额外转换，避免耗损
    - 文件读取fs.createReadStream(path,opts)中 highWaterMark大小对性能有两点影响
      - 对内存的分配和使用有一定影响
      - 设置过小，可能导致系统调用次数过多，当读取一个相同的大文件时，highWaterMark越大读取越快