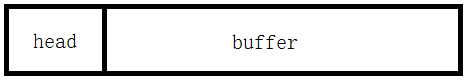
1.包结构

Tcp流的分包（第一层包）

：head消息长度包含head自身长度

：buffer 消息内容



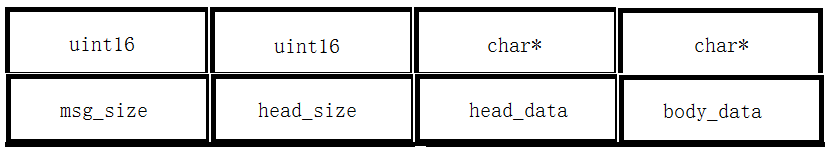
分包具体的结构（第二层包）

: msg\_size 整个消息包的长度

: head\_size 包内容简介的长度.摘要信息用于在不拆全包的情况下快速识别真实包体功能

: head\_data 包内容简介内容

: body\_data 应用层包的包体(如protobuf的序列化数据)



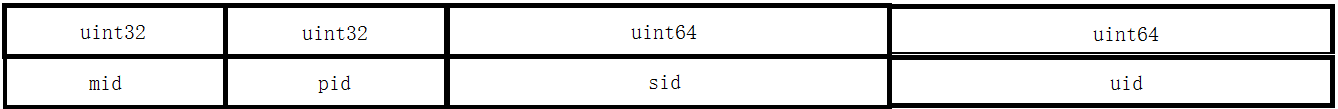
包内容简介的结构

: mid 消息号应用层包的快速

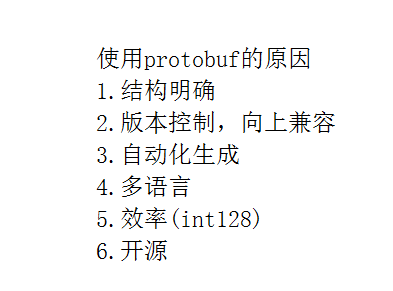
: pid 代理号当前包号是由哪个大厅发到工作代理的，用于包返回时能找到大厅

: sid 会话号大厅上的编号，表示当前用户连接在哪个大厅，在大厅的会话fd

: uid 用户id用户编号

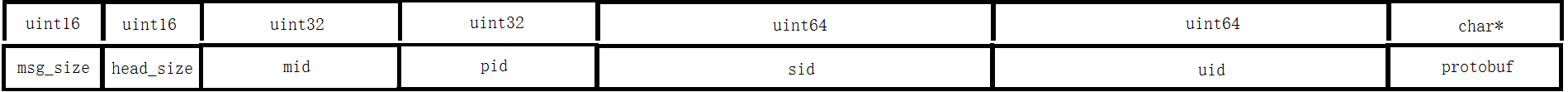


Protobuf(第三层包，当前使用的protobuf,也可以使用其他结构)



一个包的完整结构

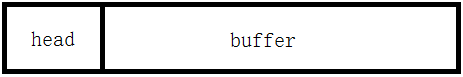
一个三层包的详细结构



2.流结构

: 一个包在流中的结构可以表示为

: 一个包头和包体



消息组成流后呈现的结构

: 每个包长度并不一致

: 而且tcp的特性是流.



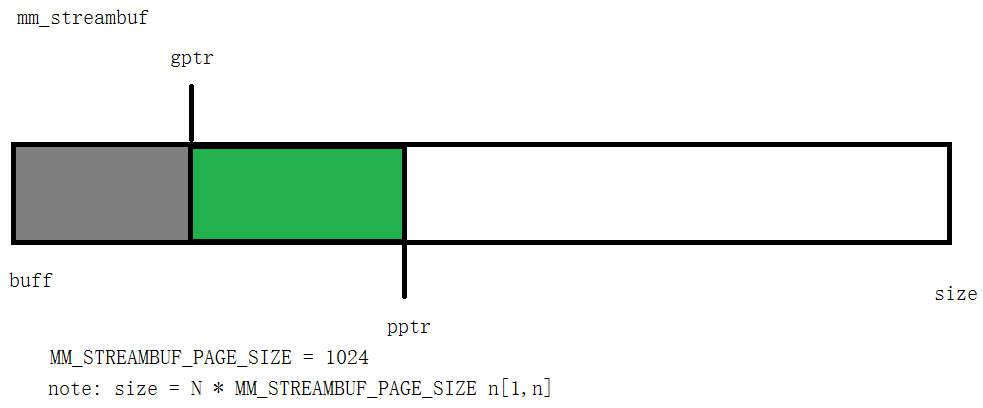
3.组流和分包

: 组流需要一个策略用以缓存从socket中读取的数据.并将其往后附加.

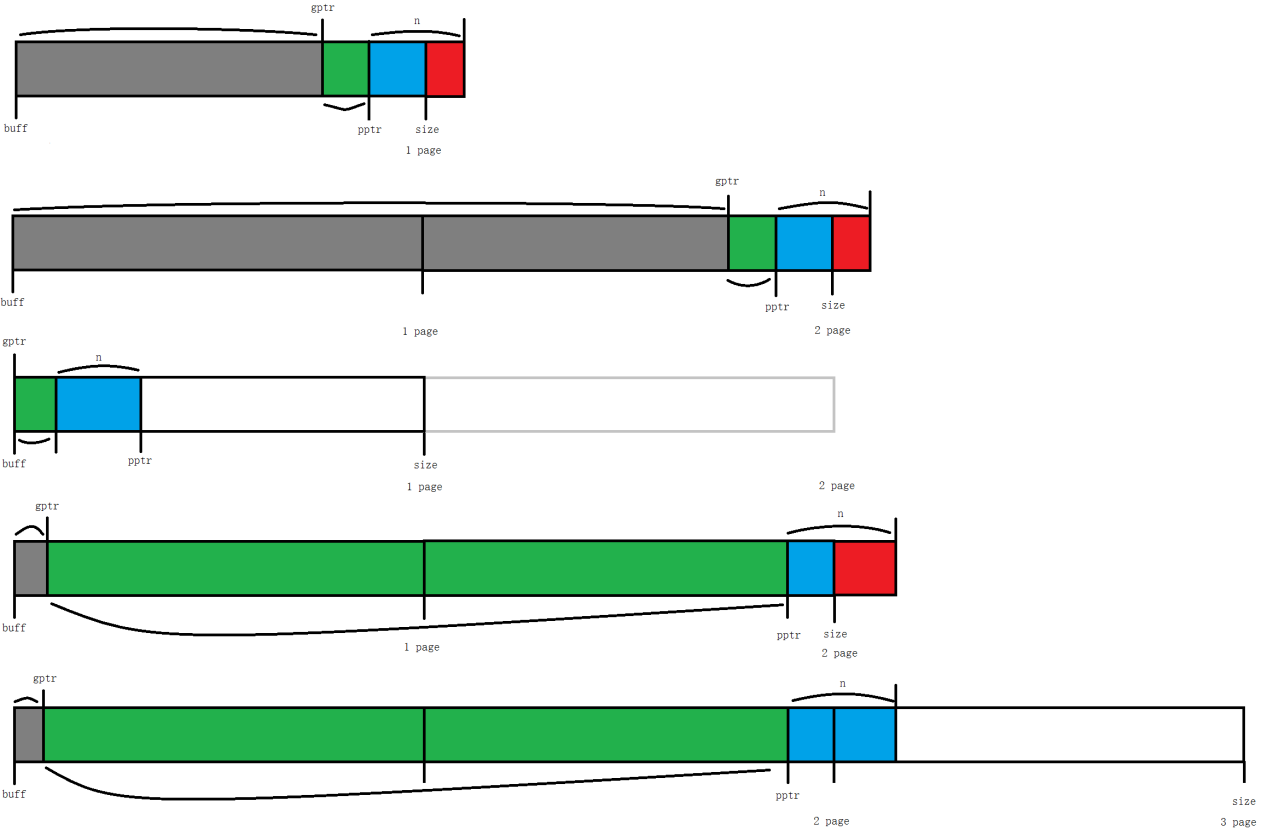
: 有策略用以将已经读取的内存进行重用.

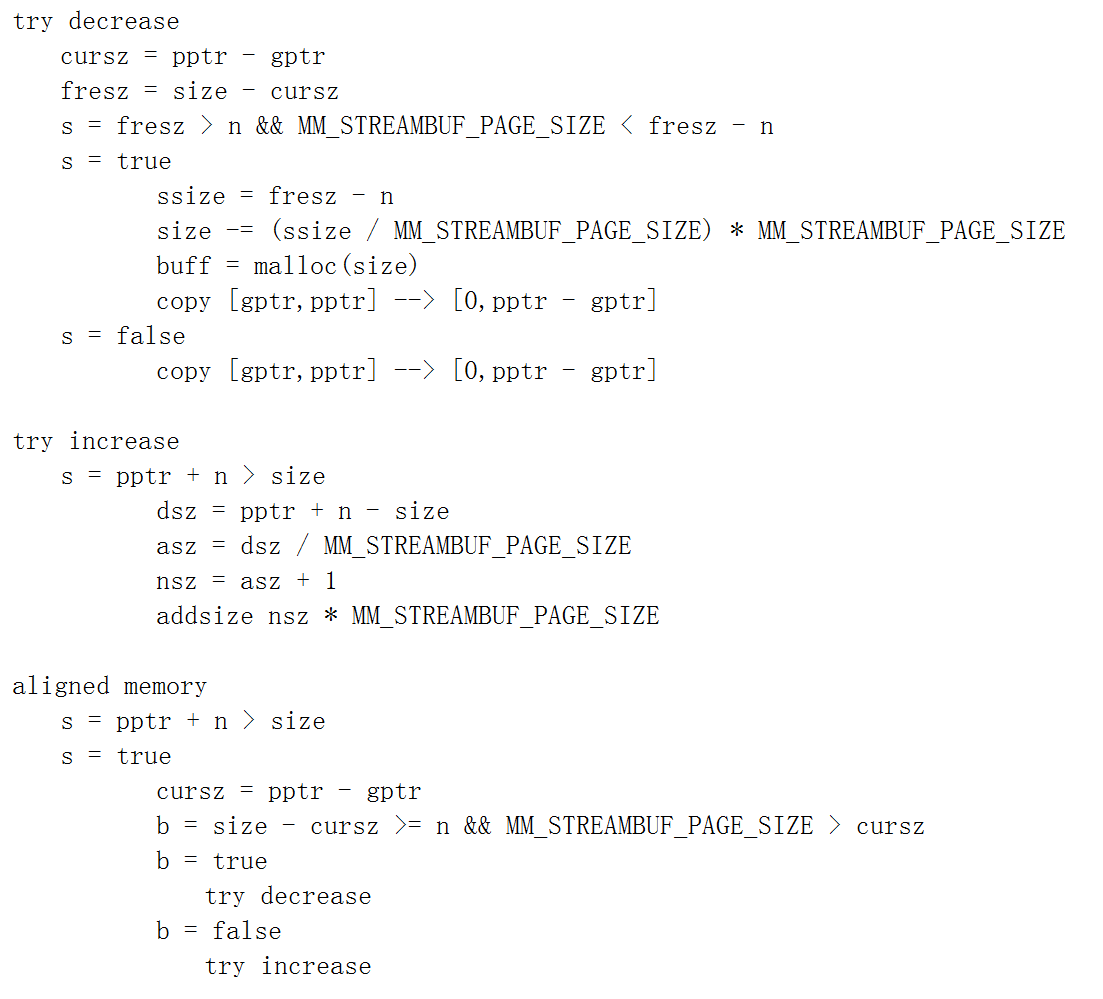
使用流缓存来实现组流mm\_streambuf

结构为:



: 数据读取之后的缓存(迁移减容)和(不足扩容)





try decrease

cursz = pptr - gptr

fresz = size - cursz

s = fresz > n && MM\_STREAMBUF\_PAGE\_SIZE < fresz - n

s = true

ssize = fresz - n

size -= (ssize / MM\_STREAMBUF\_PAGE\_SIZE) \* MM\_STREAMBUF\_PAGE\_SIZE

buff = malloc(size)

copy [gptr,pptr] --> [0,pptr - gptr]

s = false

copy [gptr,pptr] --> [0,pptr - gptr]

try increase

s = pptr + n > size

dsz = pptr + n - size

asz = dsz / MM\_STREAMBUF\_PAGE\_SIZE

nsz = asz + 1

addsize nsz \* MM\_STREAMBUF\_PAGE\_SIZE

aligned memory

s = pptr + n > size

s = true

cursz = pptr - gptr

b = size - cursz >= n && MM\_STREAMBUF\_PAGE\_SIZE > cursz

b = true

try decrease

b = false

try increase

包的序列化和反序列化

: 序列化

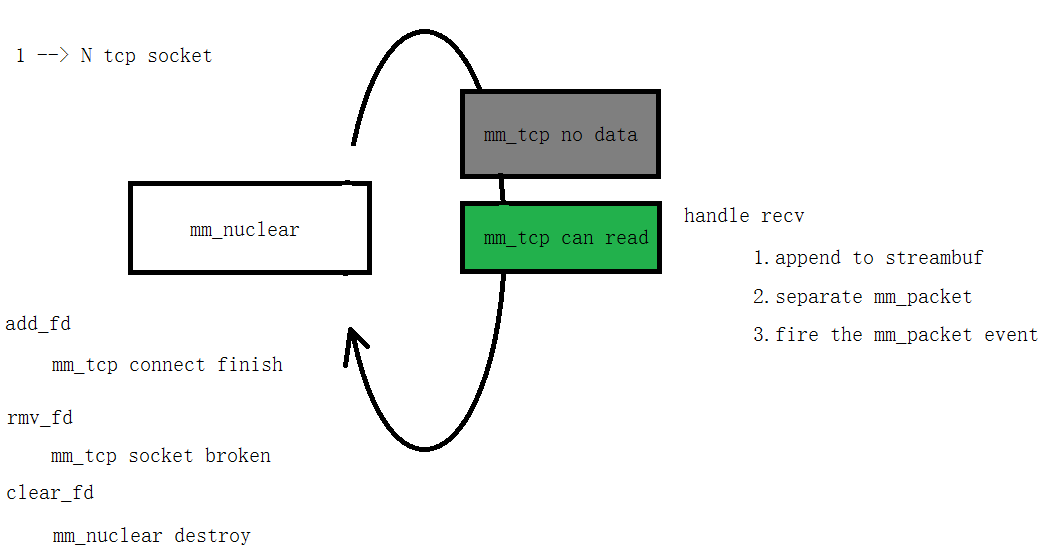
: 反序列化

以数据包的结构做流的拼装

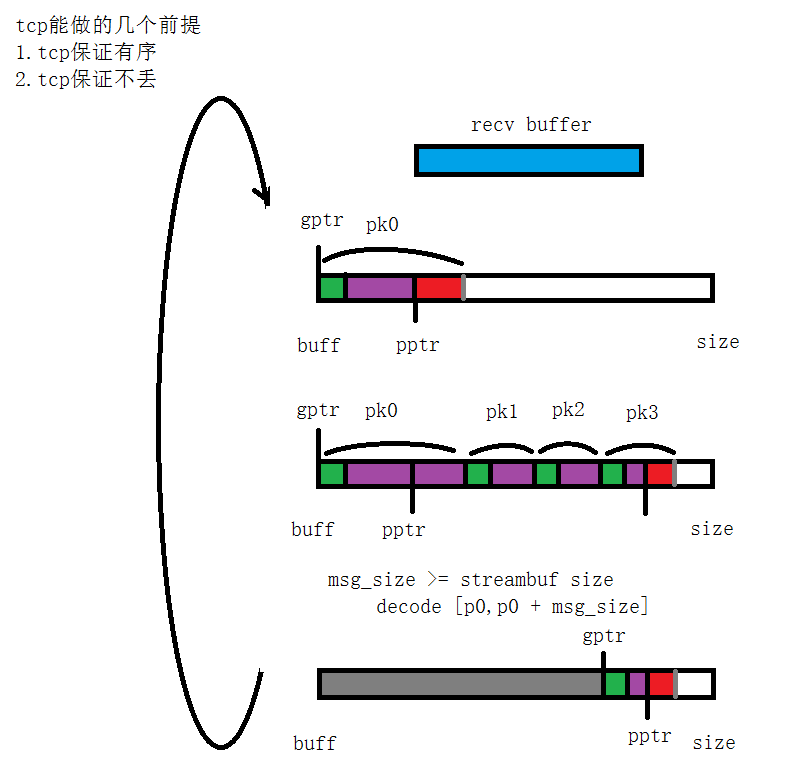
循环从流缓存中取包

网络结构

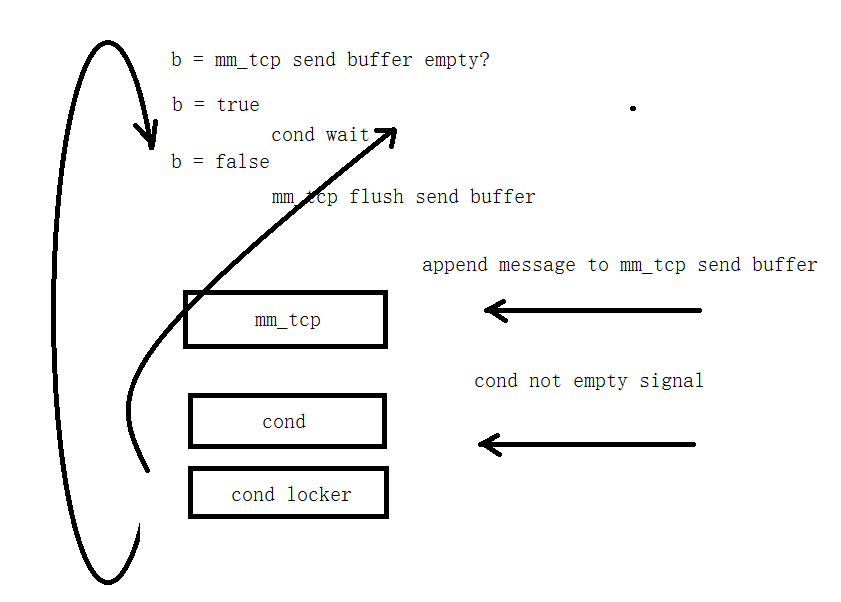
: 读写事件驱动,触发读事件,解包之后的消息回传



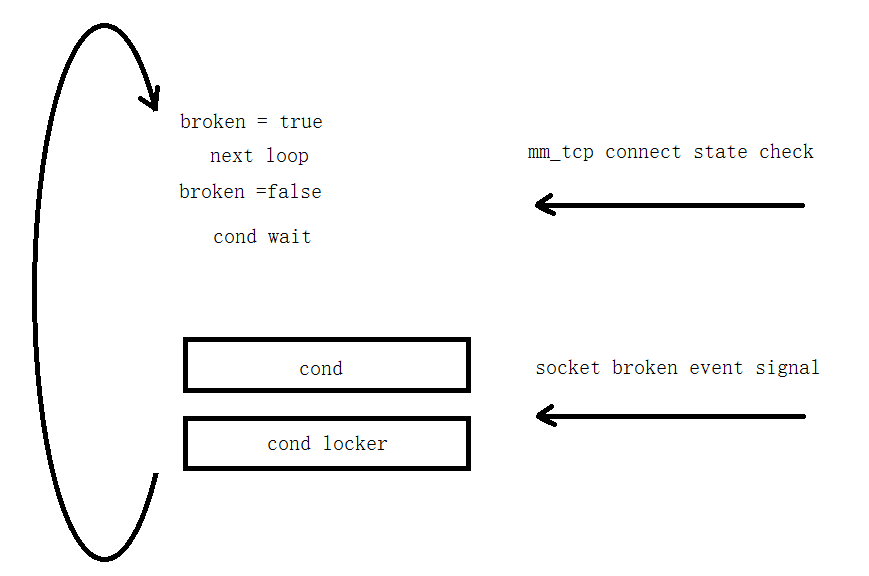
: 从接受流缓存分包



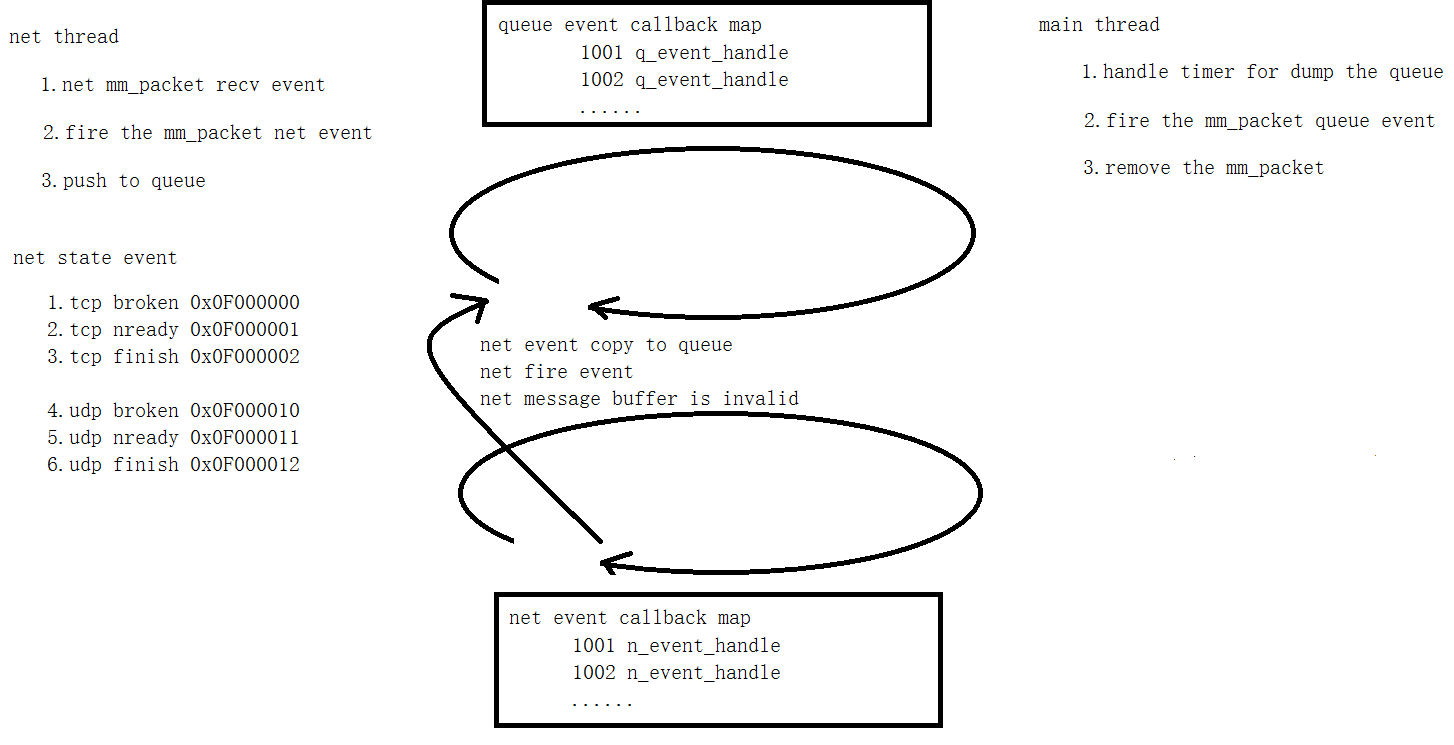
: 异步发送数据



: 异步网络状态监测



网络线程和主线程的消息派发,消息注册和回调



一个android例子

