1. 正确使用net\_send和send\_pkg\_to\_client

淘米异步框架AsyncServer提供了两个接口net\_send和send\_pkg\_to\_client来帮助应用层发送数据到对端。淘米知识库上对这两个函数的描述分别为：net\_send【父子】 send\_pkg\_to\_client【子】。表示net\_send为父子进程都会调用，send\_pkg\_to\_client只会由子进程调用。由于net\_send是开放给应用层使用的，其中一个参数是fd。这会给使用者造成一种错觉，即可以用net\_send发送数据到客户端。事实上，net\_send在具体实现的时候，是将数据发送到自己的epi结构体保存的fd中的，这些fd是子进程在自己的运行过程中通过AsyncServer提供的函数connect\_to\_svr/service调用创建的fd集合。而客户端的连接是由父进程建立的，父进程只是在子进程与客户端之间起到一个传递者的作用，他们之间的桥梁为fdsession\_t的结构，这个结构体中包含一个由父进程创建的fd，如果子进程使用net\_send来给这个fd发送数据，实际上是试图在子进程创建的fd集合中查找这个fd发送数据，如果子进程中没有这个相同的fd 数据当然不会发送成功，如果有则会发送异常数据到对端。

一般在使用的过程中，如果是给客户端发送数据一定会调用send\_pkg\_to\_client。如果是子进程发送数据到自己通过connect\_to\_svr/service连接的服务器对端，则使用net\_send发送数据。更奇葩的情况，是子进程不通过框架，自己实现了一套连接服务器或者监听端口的方法，并且自己处理这些fd 那么net\_send和send\_pkg\_to\_client都不会有效。

1. 服务器直连与proxy的作用

为什么online要直连switch, 连接db却要通过proxy？为什么热血精灵派的battle是通

过proxy连接，而D计划的battle需要直连?

在淘米的众多游戏中，大量使用了proxy来进行数据的转发，很大程度上减轻了开发需要维护各种服务器连接的负担。淘米设计proxy的初衷是为了应对DB的分表分库，他起到的作用主要是数据的路由，将online对DB的请求路由到不同的dbserver，这些dbserver可能部署在不同的物理机上。由于应用层对DB的请求是单一交互的，即只有一次req/rsp，对于一个req, DB要么给数据，要么没有数据，不会为应用层的一个req发出多个rsp。所以proxy在设计上对于req, 当他得到rsp时，这个req即被清理掉。对于DB服务，这个模式没有问题。在大多数服务模式下，比如ranking, cache 其基本操作是单一req/rsp模式的，这个模式也是没有问题的。

对于switch, 如果switch的功能仅仅是收集online与player的信息，那也可以让online通过proxy去与switch交互，而不需要直连。但是switch经常需要发一些通知包给online以及player，如果proxy与switch之间关于online/player的连接信息在switch发送rsp后被清理了，switch发送过去的数据将会在proxy那里被当成是一次新的req, 从而根据命令号重新路由，数据将会被发到未知的地方。另外，即使switch不发送通知包给online/player, 另外一个重要的问题是，如果online通过proxy连接switch, 当switch异常重启断开连接的时候，online和proxy依然正常连接，所以不知道switch断开连接了，还尝试正常发送数据，可能导致错误的回包，而且switch重启后，其收集的online/player数据都已经丢失，online也不知道是不是要继续同步这些数据给switch。所以，直连的另外一个重要原因是及时感知对端的状况。

对于上面的第二个问题，由于精灵派的对战是battle服完全控制，一个战斗在一次req/rsp中就完成，battle不会发送任何通知包给online/player。所以精灵派的battle可以通过proxy连接，而D计划的battle除了正常rsp，还需要做很多通知的操作，所以选择了直连。

事实上，因为switch上保留了online/player的信息，如果switch实现了通知包代转的功能，大部分的直连服务器也可以把通知包通过proxy发给switch,由switch再发送给online/player。这样做的消耗是数据多了2次传递，而且直连服务器需要维护与proxy的连接。