第39讲 | 知识串讲：用双十一的故事串起碎片的网络协议（下）

2018-08-15 刘超

第39讲 | 知识串讲：用双十一的故事串起碎片的网络协议（下）
朗读人：刘超　14'41'' | 6.73M

上一节，我们封装了一个长长的网络包，“大炮”准备完毕，开始发送。

发送的时候可以说是重重关隘，从手机到移动网络、互联网，还要经过多个运营商才能到达数据中心，到了数据中心就进入第二个复杂的过程，从网关到 VXLAN 隧道，到负载均衡，到 Controller 层、组合服务层、基础服务层，最终才下单入库。今天，我们就来看这最后一段过程。

7. 一座座城池一道道关，流控拥塞与重传

网络包已经组合完毕，接下来我们来看，如何经过一道道城关，到达目标公网 IP。

对于手机来讲，默认的网关在 PGW 上。在移动网络里面，从手机到 SGW，到 PGW 是有一条隧道的。在这条隧道里面，会将上面的这个包作为隧道的乘客协议放在里面，外面 SGW 和 PGW 在核心网机房的 IP 地址。网络包直到 PGW（PGW 是隧道的另一端）才将里面的包解出来，转发到外部网络。

所以，从手机发送出来的时候，网络包的结构为：

- 源 MAC：手机也即 UE 的 MAC；
- 目标 MAC：网关 PGW 上面的隧道端点的 MAC；
- 源 IP：UE 的 IP 地址；
- 目标 IP：SLB 的公网 IP 地址。

进入隧道之后，要封装外层的网络地址，因而网络包的格式为：

- 外层源 MAC：E-NodeB 的 MAC；
- 外层目标 MAC：SGW 的 MAC；
- 外层源 IP：E-NodeB 的 IP；
- 外层目标 IP：SGW 的 IP；
- 内层源 MAC：手机也即 UE 的 MAC；
- 内层目标 MAC：网关 PGW 上面的隧道端点的 MAC；
- 内层源 IP：UE 的 IP 地址；
- 内层目标 IP：SLB 的公网 IP 地址。

当隧道在 SGW 的时候，切换了一个隧道，会从 SGW 到 PGW 的隧道，因而网络包的格式为：

- 外层源 MAC：SGW 的 MAC；
- 外层目标 MAC：PGW 的 MAC；
- 外层源 IP：SGW 的 IP；
- 外层目标 IP：PGW 的 IP；
- 内层源 MAC：手机也即 UE 的 MAC；
- 内层目标 MAC：网关 PGW 上面的隧道端点的 MAC；

- 内层源 IP：UE 的 IP 地址；
- 内层目标 IP：SLB 的公网 IP 地址。

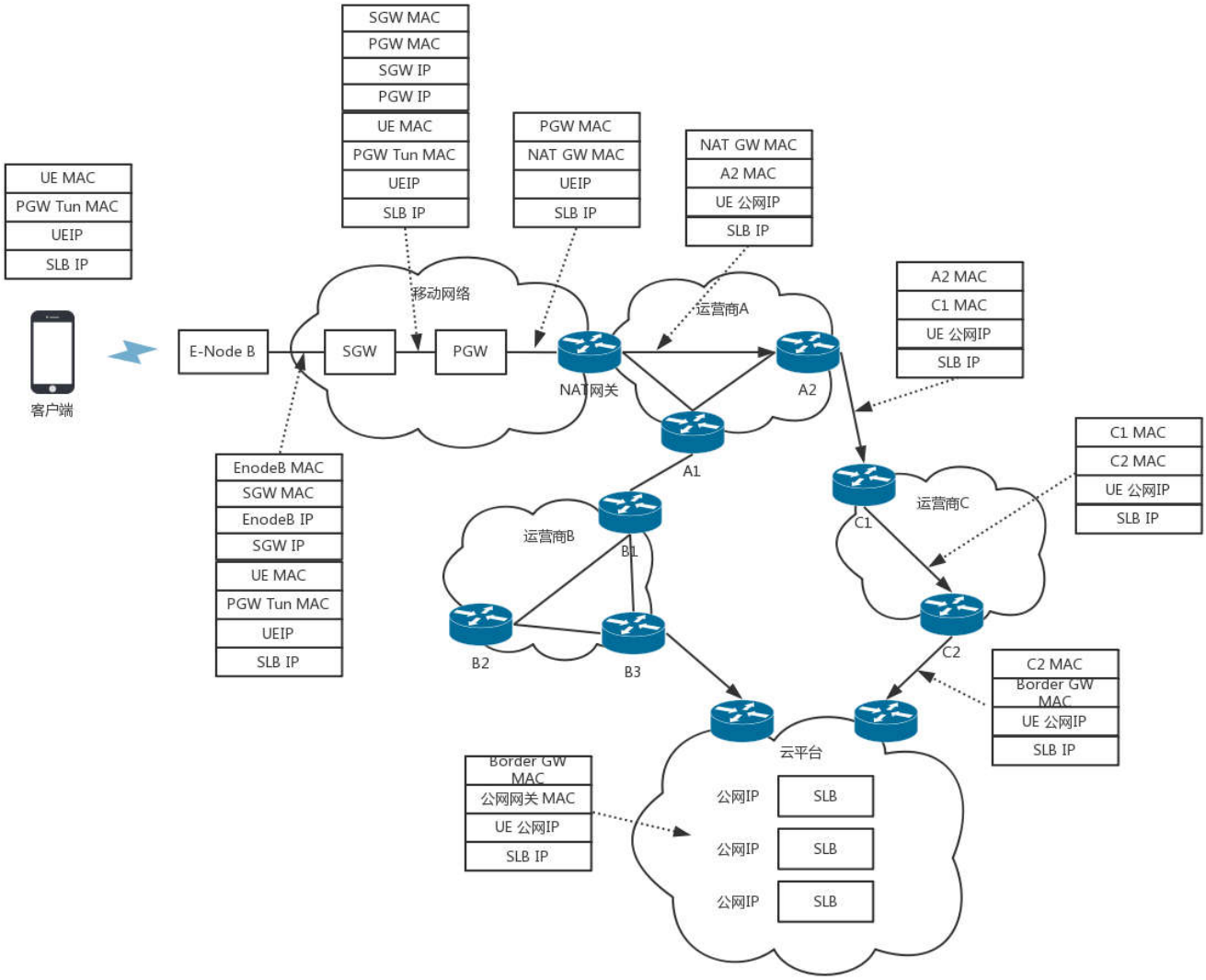
在 PGW 的隧道端点将包解出来，转发出去的时候，一般在 PGW 出外部网络的路由器上，会部署 NAT 服务，将手机的 IP 地址转换为公网 IP 地址，当请求返回的时候，再 NAT 回来。

因而在 PGW 之后，相当于做了一次欧洲十国游型的转发，网络包的格式为：

- 源 MAC：PGW 出口的 MAC；
- 目标 MAC：NAT 网关的 MAC；
- 源 IP：UE 的 IP 地址；
- 目标 IP：SLB 的公网 IP 地址。

在 NAT 网关，相当于做了一次玄奘西游型的转发，网络包的格式变成：

- 源 MAC：NAT 网关的 MAC；
- 目标 MAC：A2 路由器的 MAC；
- 源 IP：UE 的公网 IP 地址；
- 目标 IP：SLB 的公网 IP 地址。



出了 NAT 网关，就从核心网到达了互联网。在网络世界，每一个运营商的网络成为自治系统 AS。每个自治系统都有边界路由器，通过它和外面的世界建立联系。

对于云平台来讲，它可以被称为 Multihomed AS，有多个连接连到其他的 AS，但是大多拒绝帮其他的 AS 传输包。例如一些大公司的网络。对于运营商来说，它可以被称为 Transit AS，有多个连接连到其他的 AS，并且可以帮助其他的 AS 传输包，比如主干网。

如何从出口的运营商到达云平台的边界路由器？在路由器之间需要通过 BGP 协议实现，BGP 又分为两类，eBGP 和 iBGP。自治系统之间、边界路由器之间使用 eBGP 广播路由。内部网络也需要访问其他的自治系统。

边界路由器如何将 BGP 学习到的路由导入到内部网络呢？通过运行 iBGP，使内部的路由器能够找到到达外网目的地最好的边界路由器。

网站的 SLB 的公网 IP 地址早已经通过云平台的边界路由器，让全网都知道了。于是这个下单的网络包选择的下一跳是 A2，也即将 A2 的 MAC 地址放在目标 MAC 地址中。

到达 A2 之后，从路由表中找到下一跳是路由器 C1，于是将目标 MAC 换成 C1 的 MAC 地址。到达 C1 之后，找到下一跳是 C2，将目标 MAC 地址设置为 C2 的 MAC。到达 C2 后，找

到下一跳是云平台的边界路由器，于是将目标 MAC 设置为边界路由器的 MAC 地址。

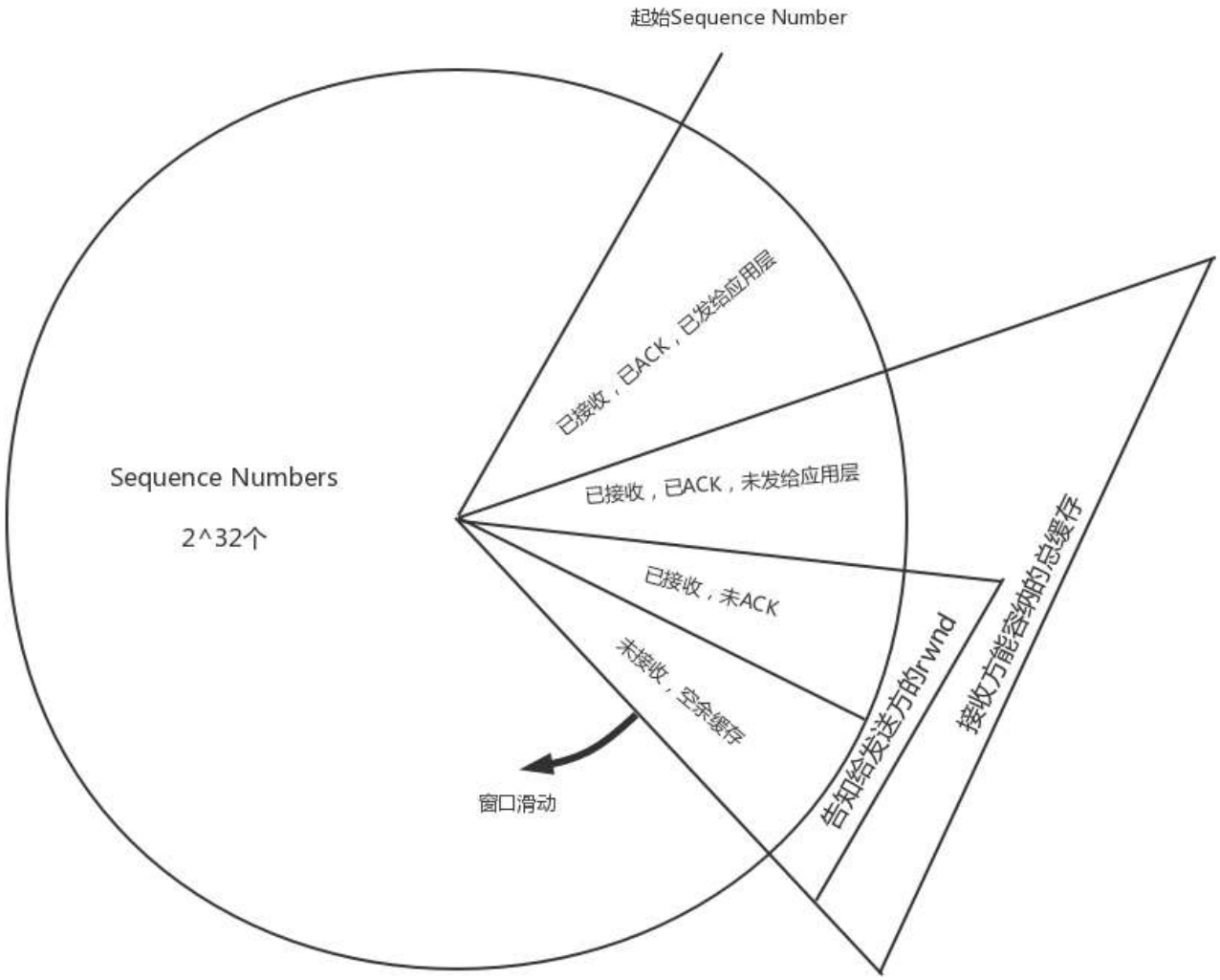
你会发现，这一路，都是只换 MAC，不换目标 IP 地址。这就是所谓下一跳的概念。

在云平台的边界路由器，会将下单的包转发进来，经过核心交换，汇聚交换，到达外网网关节点上的 SLB 的公网 IP 地址。

我们可以看到，手机到 SLB 的公网 IP，是一个端到端的连接，连接的过程发送了很多包。所有这些包，无论是 TCP 三次握手，还是 HTTPS 的密钥交换，都是要走如此复杂的过程到达 SLB 的，当然每个包走的路径不一定一致。

网络包走在这个复杂的道路上，很可能一不小心就丢了，怎么办？这就需要借助 TCP 的机制重新发送。

既然 TCP 要对包进行重传，就需要维护 Sequence Number，看哪些包到了，哪些没到，哪些需要重传，传输的速度应该控制到多少，这就是TCP 的滑动窗口协议。



整个 TCP 的发送，一开始会协商一个 Sequence Number，从这个 Sequence Number 开始，每个包都有编号。滑动窗口将接收方的网络包分成四个部分：

- 已经接收，已经 ACK，已经交给应用层的包；
- 已经接收，已经 ACK，未发送给应用层；
- 已经接收，尚未发送 ACK；
- 未接收，尚有空闲的缓存区域。

对于 TCP 层来讲，每一个包都有 ACK。ACK 需要从 SLB 回复到手机端，将上面的那个过程反向来一遍，当然路径不一定一致，可见 ACK 也不是那么轻松的事情。

如果发送方超过一定的时间没有收到 ACK，就会重新发送。只有 TCP 层 ACK 过的包，才会发给应用层，并且只会发送一份，对于下单的场景，应用层是 HTTP 层。

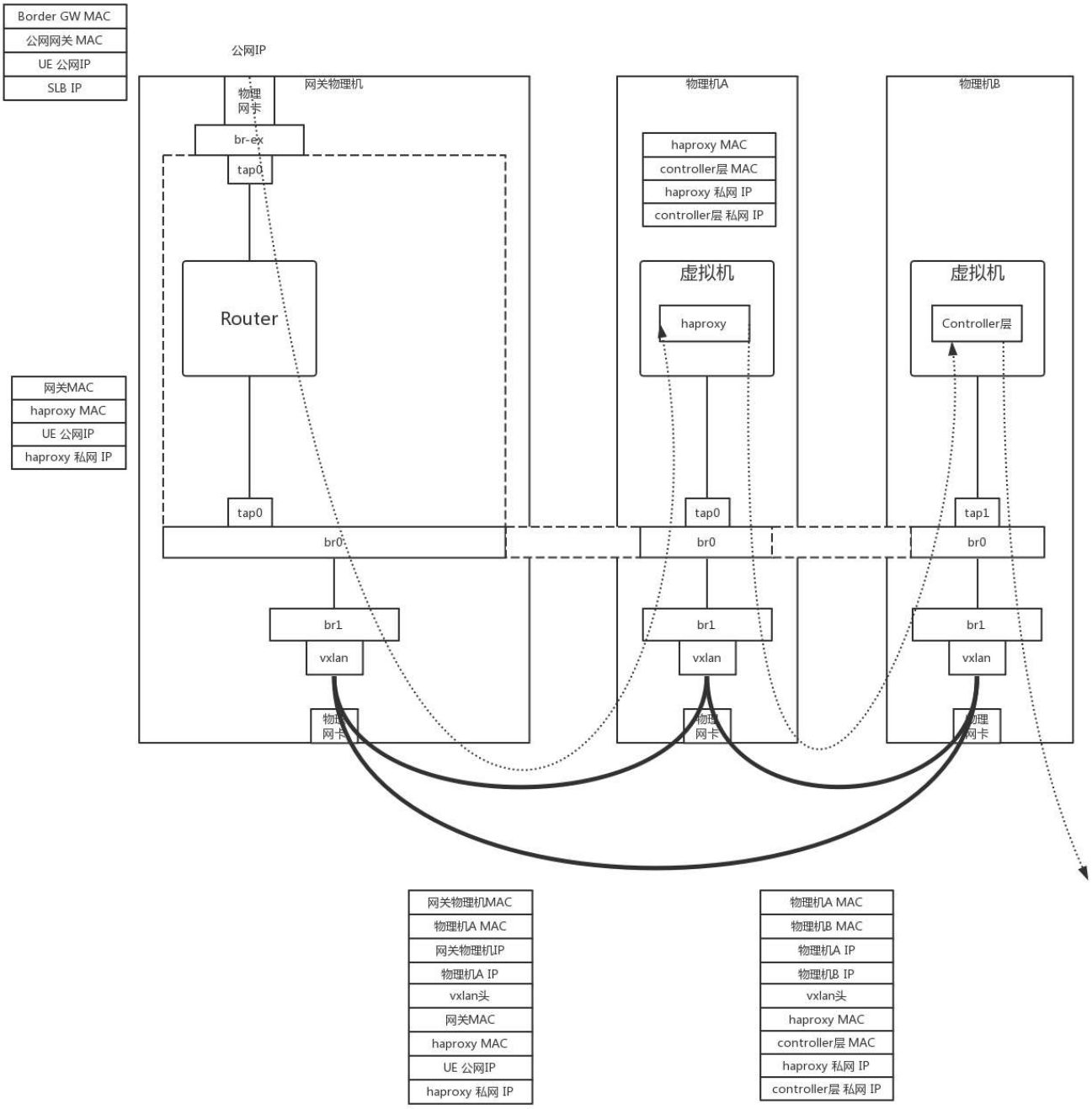
你可能会问了，TCP 老是重复发送，会不会导致一个单下了两遍？是否要求服务端实现幂等？从 TCP 的机制来看，是不会的。只有收不到 ACK 的包才会重复发，发到接收端，在窗口里面只保存一份，所以在同一个 TCP 连接中，不用担心重传导致二次下单。

但是 TCP 连接会因为某种原因断了，例如手机信号不好，这个时候手机把所有的动作重新做一遍，建立一个新的 TCP 连接，在 HTTP 层调用两次 RESTful API。这个时候可能会导致两遍下单的情况，因而 RESTful API 需要实现幂等。

当 ACK 过的包发给应用层之后，TCP 层的缓存就空了出来，这会导致上面图中的大三角，也即接收方能够容纳的总缓存，整体顺时针滑动。小的三角形，也即接收方告知发送方的窗口总大小，也即还没有完全确认收到的缓存大小，如果把这些填满了，就不能再发了，因为没确认收到，所以一个都不能扔。

8. 从数据中心进网关，公网 NAT 成私网

包从手机端经历千难万险，终于到了 SLB 的公网 IP 所在的公网网口。由于匹配上了 MAC 地址和 IP 地址，因而将网络包收了进来。



在虚拟网关节点的外网网口上，会有一个 NAT 规则，将公网 IP 地址转换为 VPC 里面的私网 IP 地址，这个私网 IP 地址就是 SLB 的 HAProxy 所在的虚拟机的私网 IP 地址。

当然为了承载比较大的吞吐量，虚拟网关节点会有多个，物理网络会将流量分发到不同的虚拟网关节点。同样 HAProxy 也会是一个大的集群，虚拟网关会选择某个负载均衡节点，将某个请求分发给它，负载均衡之后是 Controller 层，也是部署在虚拟机里面的。

当网络包里面的目标 IP 变成私有 IP 地址之后，虚拟路由会查找路由规则，将网络包从下方的私网网口发出来。这个时候包的格式为：

- 源 MAC：网关 MAC；
- 目标 MAC：HAProxy 虚拟机的 MAC；

- 源 IP：UE 的公网 IP；
- 目标 IP：HAProxy 虚拟机的私网 IP。

9. 进入隧道打标签，RPC 远程调用下单

在虚拟路由节点上，也会有 OVS，将网络包封装在 VXLAN 隧道里面，VXLAN ID 就是给你的租户创建 VPC 的时候分配的。包的格式为：

- 外层源 MAC：网关物理机 MAC；
- 外层目标 MAC：物理机 A 的 MAC；
- 外层源 IP：网关物理机 IP；
- 外层目标 IP：物理机 A 的 IP；
- 内层源 MAC：网关 MAC；
- 内层目标 MAC：HAProxy 虚拟机的 MAC；
- 内层源 IP：UE 的公网 IP；
- 内层目标 IP：HAProxy 虚拟机的私网 IP。

在物理机 A 上，OVS 会将包从 VXLAN 隧道里面解出来，发给 HAProxy 所在的虚拟机。HAProxy 所在的虚拟机发现 MAC 地址匹配，目标 IP 地址匹配，就根据 TCP 端口，将包发给 HAProxy 进程，因为 HAProxy 是在监听这个 TCP 端口的。因而 HAProxy 就是这个 TCP 连接的服务端，客户端是手机。对于 TCP 的连接状态、滑动窗口等，都是在 HAProxy 上维护的。

在这里 HAProxy 是一个四层负载均衡，也即它只解析到 TCP 层，里面的 HTTP 协议它不关心，就将请求转发给后端的多个 Controller 层的一个。

HAProxy 发出去的网络包就认为 HAProxy 是客户端了，看不到手机端了。网络包格式如下：

- 源 MAC：HAProxy 所在虚拟机的 MAC；
- 目标 MAC：Controller 层所在虚拟机的 MAC；
- 源 IP：HAProxy 所在虚拟机的私网 IP；
- 目标 IP：Controller 层所在虚拟机的私网 IP。

当然这个包发出去之后，还是会被物理机上的 OVS 放入 VXLAN 隧道里面，网络包格式为：

- 外层源 MAC：物理机 A 的 MAC；

- 外层目标 MAC：物理机 B 的 MAC；
- 外层源 IP：物理机 A 的 IP；
- 外层目标 IP：物理机 B 的 IP；
- 内层源 MAC：HAProxy 所在虚拟机的 MAC；
- 内层目标 MAC：Controller 层所在虚拟机的 MAC；
- 内层源 IP：HAProxy 所在虚拟机的私网 IP；
- 内层目标 IP：Controller 层所在虚拟机的私网 IP。

在物理机 B 上，OVS 会将包从 VXLAN 隧道里面解出来，发给 Controller 层所在的虚拟机。Controller 层所在的虚拟机发现 MAC 地址匹配，目标 IP 地址匹配，就根据 TCP 端口，将包发给 Controller 层的进程，因为它在监听这个 TCP 端口。

在 HAProxy 和 Controller 层之间，维护一个 TCP 的连接。

Controller 层收到包之后，它是关心 HTTP 里面是什么的，于是解开 HTTP 的包，发现是一个 POST 请求，内容是下单购买一个课程。

10. 下单扣减库存优惠券，数据入库返回成功

下单是一个复杂的过程，因而往往在组合服务层会有一个专门管理下单的服务，Controller 层会通过 RPC 调用这个组合服务层。

假设我们使用的是 Dubbo，则 Controller 层需要读取注册中心，将下单服务的进程列表拿出来，选出一个来调用。

Dubbo 中默认的 RPC 协议是 Hessian2。Hessian2 将下单的远程调用序列化为二进制进行传输。

Netty 是一个非阻塞的基于事件的网络传输框架。Controller 层和下单服务之间，使用了 Netty 的网络传输框架。有了 Netty，就不用自己编写复杂的异步 Socket 程序了。Netty 使用的方式，就是咱们讲Socket 编程的时候，一个项目组支撑多个项目（IO 多路复用，从派人盯着到有事通知）这种方式。

Netty 还是工作在 Socket 这一层的，发送的网络包还是基于 TCP 的。在 TCP 的下层，还是需要封装上 IP 头和 MAC 头。如果跨物理机通信，还是需要封装的外层的 VXLAN 隧道里面。当然底层的这些封装，Netty 都不感知，它只要做好它的异步通信即可。

在 Netty 的服务端，也即下单服务中，收到请求后，先用 Hessian2 的格式进行解压缩。然后将请求分发到线程中进行处理，在线程中，会调用下单的业务逻辑。

下单的业务逻辑比较复杂，往往要调用基础服务层里面的库存服务、优惠券服务等，将多个服务调用完毕，才算下单成功。下单服务调用库存服务和优惠券服务，也是通过 Dubbo 的框架，通过注册中心拿到库存服务和优惠券服务的列表，然后选一个调用。

调用的时候，统一使用 Hessian2 进行序列化，使用 Netty 进行传输，底层如果跨物理机，仍然需要通过 VXLAN 的封装和解封装。

咱们以库存为例子的时候，讲述过幂等的接口实现的问题。因为如果扣减库存，仅仅是谁调用谁减一。这样存在的问题是，如果扣减库存因为一次调用失败，而多次调用，这里指的不是 TCP 多次重试，而是应用层调用的多次重试，就会存在库存扣减多次的情况。

这里常用的方法是，使用乐观锁（Compare and Set，简称 CAS）。CAS 要考虑三个方面，当前的库存数、预期原来的库存数和版本，以及新的库存数。在操作之前，查询出原来的库存数和版本，真正扣减库存的时候，判断如果当前库存的值与预期原值和版本相匹配，则将库存值更新为新值，否则不做任何操作。

这是一种基于状态而非基于动作的设计，符合 RESTful 的架构设计原则。这样的设计有利于高并发场景。当多个线程尝试使用 CAS 同时更新同一个变量时，只有其中一个线程能更新变量的值，而其它线程都失败，失败的线程并不会被挂起，而是被告知这次竞争中失败，并可以再次尝试。

最终，当下单更新到分布式数据库中之后，整个下单过程才算真正告一段落。

好了，经过了十个过程，下单终于成功了，你是否对这个过程了如指掌了呢？如果发现对哪些细节比较模糊，可以回去看一下相应的章节，相信会有更加深入的理解。

到此，我带着你用下单过程把网络协议的知识都复习了一遍。授人以鱼不如授人以渔。下一节，我将会带你来搭建一个网络实验环境，配合实验来说明理论。

欢迎你留言和我讨论。趣谈网络协议，我们下期见！

精选留言

程启 0

超哥，赞一个！

有个小问题，下单系列一和二中，系列一里面说拿到三个slb的公网ip，二里面直接就建连了。
本意是说客户端localdns随机返回一个，还是slb自身做负载均衡提供三个里面的一个？

2018-08-15

程启 0

牛

2018-08-15

favorlm 0

忘记vxlan是什么了

2018-08-15