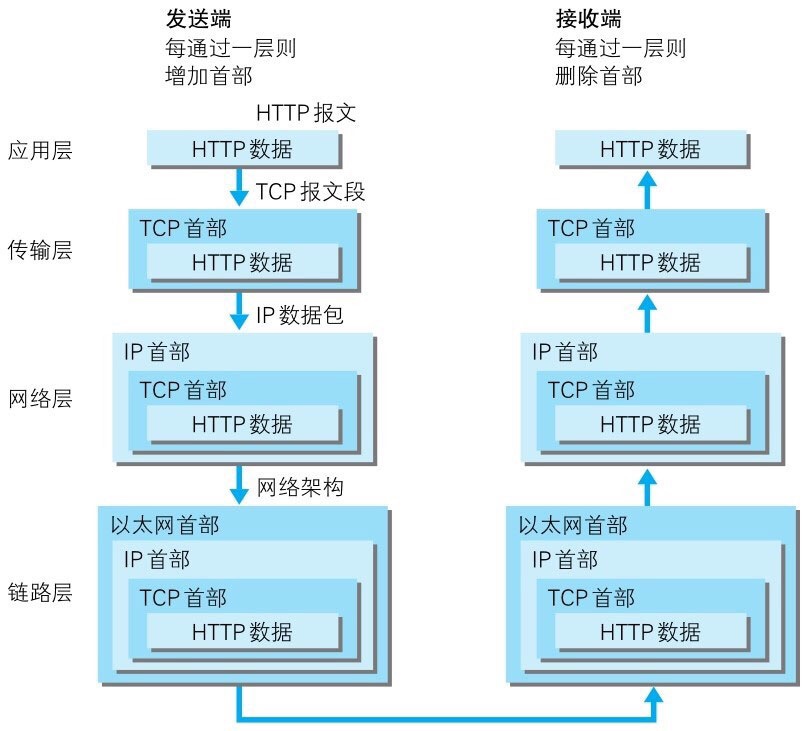
从前端开始的全链路优化

1. 从用户输入一个url开始，到渲染出整个页面，共经历了哪些内容？
   1. 输入一个url， 发送了http请求；
   2. DNS会将http请求里面的域名，解析成对应的ip地址；
   3. http请求数据->运输层（tcp+udp）用来把http的内容进行分割，加上序号和端口号，把内容安全的进行发送（相当于把自行车拆成一个个零件发送）；并且加上tcp的报文头部；
   4. 请求数据到达网络层（ip地址+mac地址）相当于把内容放到准确的地方，然后再加上ip的报文头部，（ip地址相当于小区，mac地址相当于门牌号）到达数据链路层；
   5. 接收端的服务器在链路层接收到数据，按序往上层发送，一直到应用层，才能真正算接收到客户端发来的请求。
   6. 

2、决定网络通信的两个方面：延迟与带宽。

延迟： 分组从信息源发送到目的地所需的时间。

带宽：逻辑或物理通信路径最大的吞吐量；

一、HTTP优化

1. HTTP中有一种称为内容编码的功能。指应用在实体内容上的编码格式，并保持实体原样压缩。内容压缩后实体由客户端解压并负责解码。Gzip、compress
2. 发送大容量数据时，把数据分割成多块，能够让浏览器逐步显示。
3. 通信数据转发程序：
   1. 代理（缓存代理、透明代理）
      1. 利用缓存技术，减少网络带宽流量；
      2. 组织内部针对特定网站的访问控制，以获取访问日志为主要目的；
   2. 网关
      1. 可以提高通信安全性；
   3. 隧道
      1. 使用SSL加密，保证客户端与服务端安全的通信；
   4. 反向代理
      1. 客户端向反向代理服务端发送请求，接着反向代理将判断向何处（源服务器）转交请求，并将获得的内容转交给客户端；
      2. 负载均衡：源服务器可能有多台，那么反向代理就会进行轮询、加权轮询等向源服务区转交请求；
4. DSN优化：

DNS被优化原因：DNS是应用层的协议，每次解析一次需要20-120mg。

DNS优化方式：

1. 减少DNS请求次数；
2. 进行DNS预获取；

DNS Prefetch，即DNS预获取。DNS Prefetching 是让具有此属性的域名不需要用户点击链接就在后台解析，而域名解析和内容载入是串行的网络操作，所以这个方式能 减少用户的等待时间，提升用户体验 。

参考地址：

<http://skyhome.cn/div_css/301.html>

<http://blog.csdn.net/Bin_Going/article/details/72858061>

<https://www.cnblogs.com/rongfengliang/p/5601770.html>

三、TCP/IP：

TCP/IP优化

1、最初的TCP在连接建立成功后会向网络中发送大量的数据包，这样很容易导致网络中路由器缓存空间耗尽，从而发生拥塞。

2、三次握手带来的延迟使得每次创建一个新TCP连接都要付出巨大代价，所以这里是提升TCP应用性能的关键。

解决第一点：慢启动；

原理：新建立的连接不能够一开始就发送大尺寸的数据包，而只能从一个小尺寸的包开始发送，在发送和数据被对方确认的过程中去计算对方的接收速度，来逐步增加每次发送的数据量（最后到达一个稳定的值，进入高速传输阶段。相应的，慢启动过程中，TCP通道处在低速传输阶段），以避免上述现象的发生。这个策略就是慢启动。

解决第二点：

Nagle算法：

任意时刻，最多只能有一个未被确认（发出去没有收到ACK确认该数据已收到）的小段（小于MSS尺寸的段）。

原理：

为了尽可能发送大块数据，避免网络中充斥着许多小数据块。

ACK延迟机制：

当Server端收到数据之后，它并不会马上向client端发送ACK，而是会将ACK的发送延迟一段时间（假设为t），它希望在t时间内server端会向client端发送应答数据，这样ACK就能够和应答数据一起发送，就像是应答数据捎带着ACK过去。

造成的问题：

如果一个 TCP 连接的一端启用了Nagle算法，而另一端启用了ACK延时机制，而发送的数据包又比较小，则可能会出现这样的情况：发送端在等待接收端对上一个packet的Ack才发送当前的packet，而接收端则正好延迟了此Ack的发送，那么这个正要被发送的packet就会同样被延迟。当然Delayed Ack是有个超时机制的，而默认的超时正好就是40ms。

问题：现代的 TCP/IP 协议栈实现，默认几乎都启用了这两个功能，那岂不每次都会触发这个延迟问题？

解决Nagle和ACK延迟机制的办法：

1. 优化协议

连续 write 小数据包，然后 read 其实是一个不好的网络编程模式，这样的连续 write 其实应该在应用层合并成一次 write。

1. 开启TCP\_NODELAY

简单地说，这个选项的作用就是禁用Nagle算法，禁止后当然就不会有它引起的一系列问题了。使用setsockopt可以做到：

1. **static** **void** \_set\_tcp\_nodelay(**int** fd) {
2. **int** enable = 1;
3. setsockopt(fd, IPPROTO\_TCP, TCP\_NODELAY, (**void**\*)&enable, **sizeof**(enable));
4. }

参考url：

<http://blog.csdn.net/fred1653/article/details/51689617>

<http://blog.csdn.net/majianfei1023/article/details/51558941>

慢启动：

<http://blog.chinaunix.net/uid-29783732-id-5052329.html>

<http://blog.csdn.net/pud_zha/article/details/7919374>

四、提高http连接性能：

http管道化：并行连接

持久化连接

管道化优势：HTTP管线化可以克服同域并行请求限制带来的阻塞，它是建立在持久连接之上，是把所有请求一并发给服务器，但是服务器需要按照顺序一个一个响应，而不是等到一个响应回来才能发下一个请求，这样就节省了很多请求到服务器的时间。

劣势：HTTP管线化仍旧有阻塞的问题，若上一响应迟迟不回，后面的响应都会被阻塞到。

HTTP持久连接：建立TCP链接，只要任何一方没有断开的意思，那么就保持TCP链接状态；优势：减少了建立TCP断开TCp等连接的额外开销，减轻了服务器的负担。缺点：不是所有的服务器都支持持久链接。

参考：<https://www.2cto.com/kf/201510/446824.html>

五、缓存机制

缓存的方式：

localStorage

sessionStorage

cookie

资源缓存：url为id，a.{hashName}.js

session，memcache，

Redis

检测缓存是否失效的方式：

A、Expires:1、Expires的值为服务端返回的到期时间,即下一次请求时，请求时间小于服务端返回的到期时间，直接使用缓存数据。2、到期时间是由服务端生成的，但是客户端时间可能跟服务端时间有误差，这就会导致缓存命中的误差。

所以HTTP 1.1 的版本，使用Cache-Control替代。

B、Cache-Control

Cache-Control 是最重要的规则。常见的取值有private、public、no-cache、max-age，no-store，默认为private。

private: 客户端可以缓存

public: 客户端和代理服务器都可缓存（前端的同学，可以认为public和private是一样的）

max-age=xxx: 缓存的内容将在 xxx 秒后失效

no-cache: 需要使用对比缓存来验证缓存数据（后面介绍）

no-store: 所有内容都不会缓存，强制缓存，对比缓存都不会触发（对于前端开发来说，缓存越多越好，so...基本上和它说886）

C、Last-Modified：服务器在响应请求时，告诉浏览器资源的最后修改时间。

If-Modified-Since：若Last-Modified> If-Modified-Since说明资源又被改动过，则响应整片资源内容，返回状态码200；

若Last-Modified<= If-Modified-Since说明资源无新修改，则响应HTTP 304，告知浏览器继续使用所保存的cache。

D、Etag：服务器响应请求时，告诉浏览器当前资源在服务器的唯一标识（生成规则由服务器决定）。

E、If-None-Match If-None-Match === Etag（相等 304；不等相应整片资源内容）

Etag / If-None-Match（优先级高于Last-Modified / If-Modified-Since）

参考

url：<https://segmentfault.com/a/1190000007767507>

url: <https://www.cnblogs.com/chenqf/p/6386163.html>

六、流

需要流的原因：上传文件，传了一半断开了，怎么办？

所以需要断点续传程序：把一个文件分成多块，记上标记，然后往服务器传，服务器接收后，返回一个标记，然后再继续发直到把所有的都传完。

<http://blog.csdn.net/qq_29158381/article/details/50747621>

1. 数据库

1、从数据库将经常使用的、不改变的、数量小的数据从服务器中拉取出来，放到redis里面，下次在请求的时候，直接从redis里面取，会加快速度。