计算机网络—自顶向下的方法

一、从应用软件的想法到应用体系结构。

1. 如果想为人类提供伟大的服务，**如何将你的想法变成真实世界的网络应用**?

   研发网络应用程序的核心：写出能运行在不同的端系统、并通过网络彼此通信的程序。

   也就是说，并不需要在网络核心设备，如路由器或交换机上运行软件，只需要编写在多台端系统上运行的软件。

   这也就是说，网络体系结构和应用程序体系结构是两种概念。

   > 网络体系结构：是固定的，给应用程序提供特定的服务集
   > 应用体系结构：由应用程序研发者设计，规定了如何在各种端系统上组织应用程序。

2. 主流的两种应用体系结构：客户-服务器体系结构，对等P2P体系结构

   无论是客户-服务器体系结构还是P2P体系结构，都能找到一个client进程和一个server进程。（在p2p中，下载方为client，上传方为server）。又名：应用程序的客户端和服务器端。

3. **客户-服务器体系结构**：有一个总是打开的主机称为服务器，它服务于许多来自其他客户的主机的应用。一个主机回应所有的客户端会导致不堪重负，所以配备大量主机的数据中心常被用于创建更强大的服务器。

   特点：客户之间不直接通信，都是通过服务器交流。

   典型代表：Web、FTP、Telnet、电子邮件。

4. P2P体系结构：应用程序在间断连接的主机对之间使用直接通信。减小了服务器方设备和对带宽的支付。

   典型代表：文件共享，因特网电话Skype，对等方协助下载加速器（迅雷）。

二、多个端系统上的程序是如何进行通信的呢?

1. 通信的本质：进程之间的相互通信

   进程运行在相同的端系统上，可以通过进程间的通信机制进行相互通信。

   当运行在不同的端系统上的时候，通过跨越计算机网络体系结构**交换报文**，而相互通信。

   所以，**通信的本质**是通信进程对之间的信息交换。

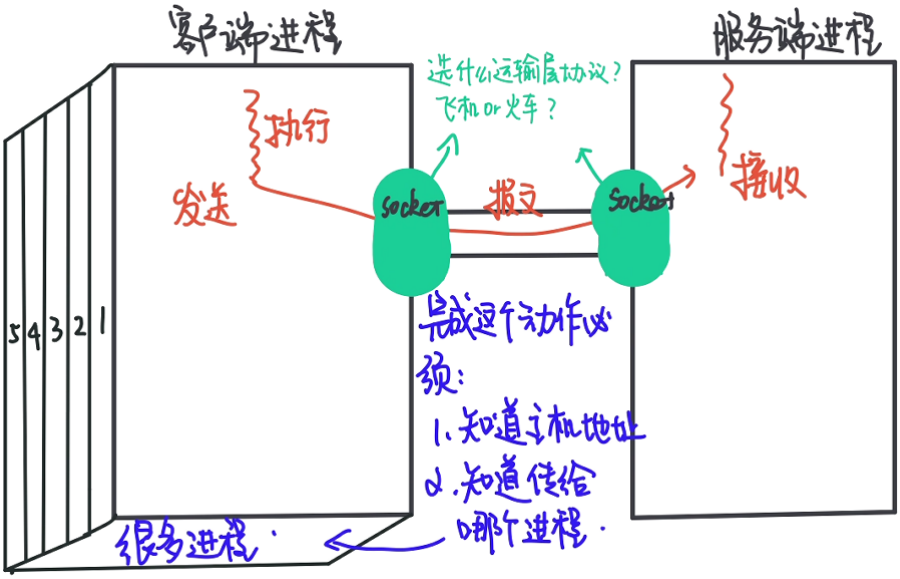
2. 通信进程对之间互相发送报文，要经过计算机网络体系结构，但是在此之前，要推开一个门，才能进入基础设施，这个门叫做套接字socket。（应用层和传输层之间的接口)。

套接字是建立在网络应用程序的**可编程接口**，也被称为**应用程序编程接口**（API）。

也就是说，应用程序开发者可以控制套接字在应用端的一切，但是对于套接字以下的部分几乎没有控制权。（可以选选传输层的协议啊，设定几个参数啊，最大缓存和最大报文长度啊，剩下就没了）。

3. **socket里面一个重要的东西—寻址**

众所周知，两个应用软件之间的通信就是两个通信进程对的通信。所以socket里面包含两个非常重要的东西： 1. 主机的地址。2. 进程的标识符。（统称为端口port）



4. **socket之下的运输服务：** 选飞机还是选火车？ 要通过应用程序的要求来决定。（TCP和UDP）

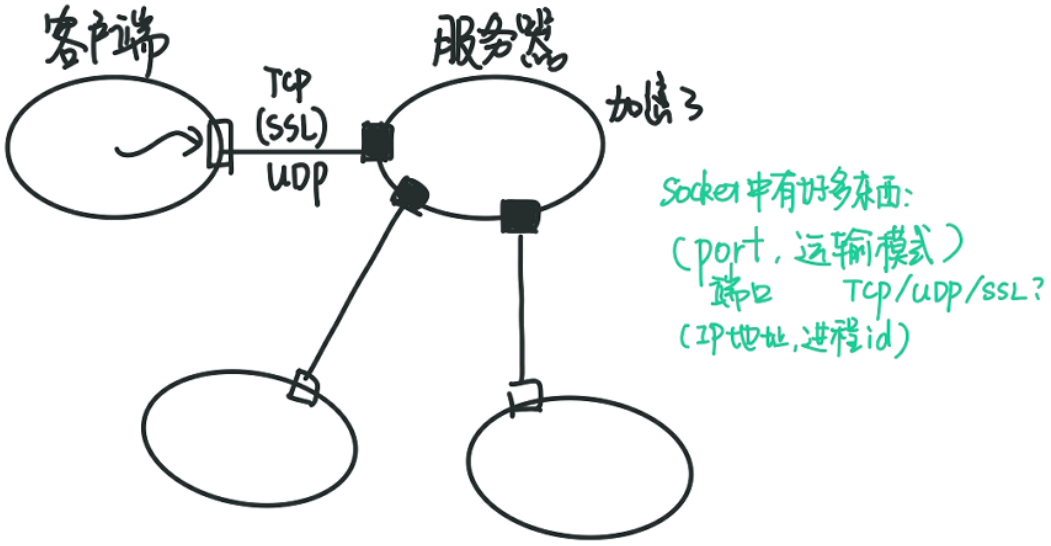
（1）选火车（**TCP服务模式**）：面向连接的服务，可靠的数据传输。

> 1.面向连接服务的本质：在应用层数据报文开始流动之前，TCP让客户和服务器相互交换运输层控制信息。也就是握手的过程（三次握手），握手结束之后，会形成一个全双工的TCP连接，当应用程序接受的时候，就应该拆除连接。
>
> 2.可靠的数据传送服务：当应用程序的一端将字节流流进socket的时候，它能够依靠TCP将相同的字节流交付给接收方的套接字。没有字节的丢失和冗余。
>
> 3. 拥塞控制机制：不一定能为通信进程带来好处，但是能给整个网络带来好处（有交警）。

典型例子：电子邮件、远程终端访问、Web、文件传输。

Notes：

SSL： 一种针对于TCP传输的加密服务。



无论是TCP还是UDP都不安全，都没有任何的加密措施。SSL是一种对于TCP的加强（包括现有的，高度优化的库和类）。SSL自己的套接字，发送进程通过SSL传递自己的明文，然后加密传送给对方的套接字。

（2）选飞机（**UDP服务模式**）：提供轻量级的运输协议，仅提供最小服务。

没有握手的过程，没有可靠，可能乱序到达，不保证一定能接收到。也没有拥塞控制。

典型例子：因特网电话

所有的运输协议都不能满足定时和带宽的保证，这取决于网络体系结构的维持，并不取决于应用体系结构。

5. **应用层协议**

   报文发送进套接字，再传入网络体系结构，从而形成进程对之间的相互通信。

   应用层协议规定了应用程序进程对之间**如何相互传递报文**：如何构造报文，报文字段含义，什么时候发送报文等等。

应用层协议只是网络应用很小的一部分，但是是很重要的一部分：

例如：Web应用=web服务器+web客户端浏览器+应用层协议（规定了客户浏览器和服务器之间的报文格式和序列）。

应用层协议的学习从例子出发：

（1）web应用层协议：超文本传输协议HTTP（端口号80），典型的客户-服务器模式。

- 客户端：浏览器（Internet Explorer 或者 Firebox）。服务器：web服务器（Apache，Microsoft Internet Information）

web页面（也叫文档）是由对象组成的，对象是一个文件，一个图形，或者一个小程序，视频片段等等。多数Web页面有一个HTML文本，和一堆对象。HTML基本文件通过对象的URL地址引用页面中的其他对象。

URL包括主机名+路径名： 例如：http://www.someSchool.edu/someDepartment/picture.gif

http://www.someSchool.edu是主机名，后面的是路径名。可以下载一个html文件看看里面有些啥。

web页面在浏览器里，web服务器（存储对象文件）的在服务器上。

什么情况下用web应用呢？ 就是你点击一个超链接，然后进入另一个界面的过程。

- **非持续连接和持续连接**

用web服务的时候，客户会发出一系列的请求，服务器对每个请求进行响应。那么每个请求可以经单独的TCP连接发送（非持续性连接），也可以将所有的请求经过相同的TCP连接发送（持续连接）。

- **当我们点击一个超链接的时候，会发生什么?**

（1）HTTP客户进程在端口号80发起一个到服务器www.someSchool.edu 的TCP连接，客户和服务器通过socket关联。

（2）三次握手

客户进程通过socket发送请求报文，包含路径名 /someDepartment/home.index

HTTP服务器接收到请求报文，通过socket向客户发送响应报文。

HTTP客户端给HTTP服务器回复。 【三次握手结束】

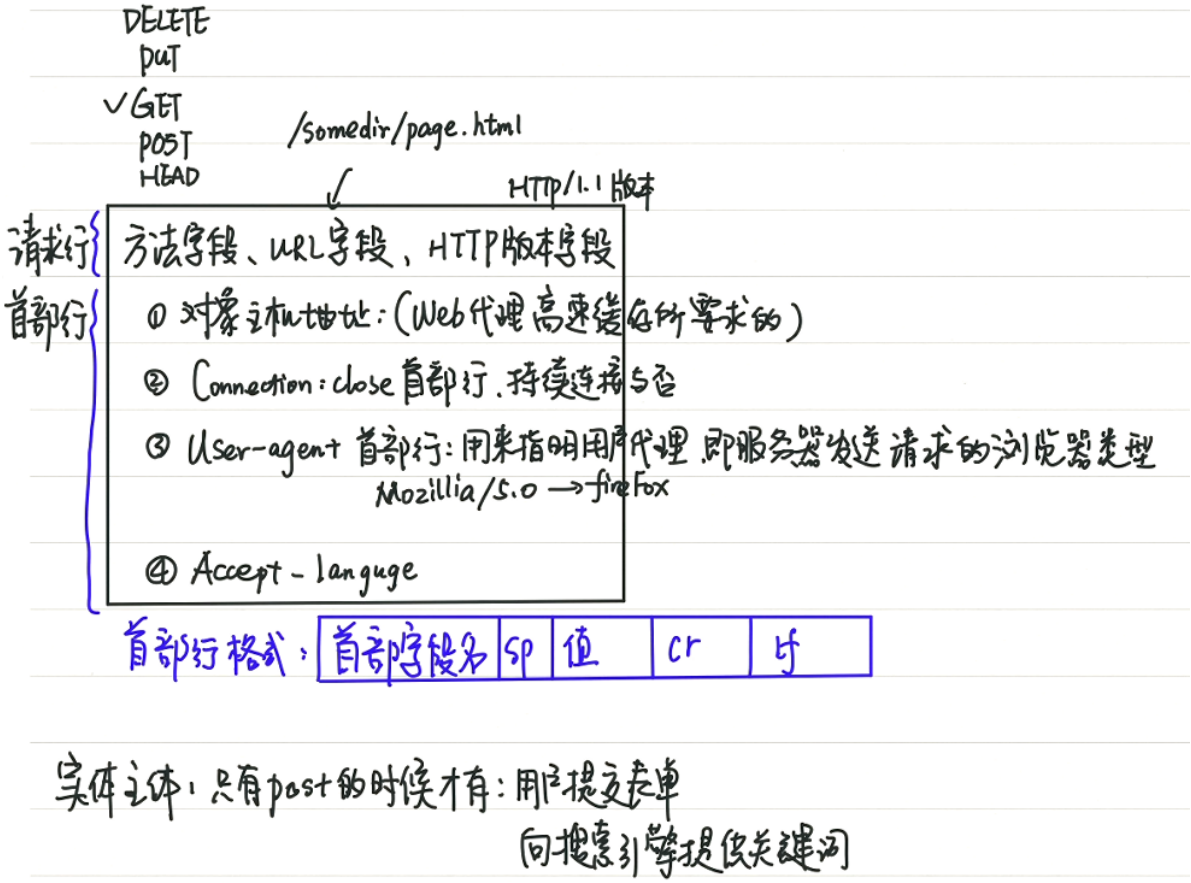
（3）发送HTML文本

（4）断开TCP连接（四次挥手）

加入这个文件里有1个HTML文件，还有10个JPEG图形，如果用单独的TCP连接，那么就会有11个TCP连接，可以设置是串行还是并行。

- **HTTP请求报文和响应报文**

1. **请求报文**



2. **响应报文**（其中有Content-Length等信息）

   状态200: 请求成功

   301: 请求的对象被转移了，新的URL定义在响应报文的Location：首部行中，客户软件将获取新的URL

   400:该请求不能被服务器理解

   404:Not found 被请求的文档不在服务器上

   505:服务器不支持请求报文的HTTP协议版本

- **用户与服务器的交互：cookie**

我们希望内容和用户联系起来。

cookie技术有4个组件：

1. HTTP响应报文中的一个cookie首部行
2. HTTP请求报文中的一个cookie首部行
3. 在用户端系统中保留一个cookie文件，并由浏览器管理
4. 位于web站点的一个后端数据库

cookie用来将web页面和用户联系起来：假设Susan已经访问过Amazon.com的eBay站点

1. Susan给Amazon Web服务器发送请求报文。
2. 该Web服务器将产生一个唯一的识别码，并作为索引在它的后端数据库中产生的一个表项。

3. 接下来，Amazon服务器用一个包含Set-cookie：首部的HTTP响应报文对Susan的浏览器进行回应，这个响应可能是cookie：1678。
4. 当Susan浏览器（客户端）收到HTTP响应文件，就会看到cookie:1678, 该浏览器就会在它管理的特定cookie文件中加一行，包含服务器主机名（亚马逊服务器）和Set-cookie：首部中的识别码。
5. 之后Susan每一次访问Amazon网站的时候，都会带着cookie码，这样浏览器就能密切跟踪Susan在Amazon站点的活动。

想想一个场景: 在理发店，有客人用自己存的洗发水，但是普通客人只能用公用的洗发水。

如果我想存自己的洗发水，我得跟人家老板说：

”我想存一个自己的洗发水“（对应步骤1）

老板把你的洗发水放进1678号隔间（对应步骤2），然后说以后1678就是你的洗发水（对应步骤3）

以后我再想去洗头，我就要和老板说，我要用1678号洗头（对应步骤4）

- **web缓存器（web cache）**也叫 **代理服务器（proxy server）**

  能够代表初始Web服务器来满足HTTP请求的网络实体。

  web缓存器有自己的磁盘存储空间，保存最近请求对象的副本。

  web缓存器是如何copy一份服务器上的副本呢?

  1. 浏览器建立一个和web缓存器的TCP连接，并向缓存器发送HTTP请求。
  2. web缓存器检查是否有该副本，如果有，web缓存直接向用户浏览器返回该对象。
  3. 如果没有该对象（证明需要copy了），就打开一个与该对象的初始服务器的TCP连接，发送对该对象的一个HTTP请求。在收到请求之后，初始服务器向该Web缓存器发送该对象的HTTP响应。传输对象文件
  4. 当Web缓存器收到该文件的时候，先自己保存一份，然后发给客户副本。

  安装web缓存器大大减少了客户请求响应时间，减轻初始服务器的压力。