**自顶向下的计算机网络：**

# 应用层

## 应用层协议原理

### 网络应用体系结构

网络的体系结构是固定的，但是网络应用的体系结构可以由程序研发者设计，有一些常用的主流架构：C/S架构，B/S架构和P2P架构。

**C/S架构（client service architecture）**：有一个总是打开的主机作为服务器，服务来自其他主机作为客户的请求。

当然一个主机可能跟不上所有客户的请求，为此需要配备大量的**数据中心（data center）**来创建虚拟服务器。也就是分布式服务

图示

描述已自动生成

**B/S架构（browser service architecture）**: 因为使用浏览器作为服务器，业务逻辑和数据操作都集中在服务器端。

图示

中度可信度描述已自动生成

**P2P体系结构（P2P architecture）**并不依赖于数据中心，而是在主机之间使用直接通信，这些主机称之为**对等方**。P2P具有**自扩展性**（self-scalability），每个对等方都因请求产生工作负载，但每个对等方向其他对等方发送文件也为系统增加了服务能力。未来P2P应用于高度非集中式结构，面临着安全性、性能和可靠性的挑战。

### 进程通信

进行通信的不是**程序program**，而是**进程process**。在两个不同的端系统上的进程，跨越计算机网络交换报文message而相互通信。发送进程向网络发送报文，接受进程接受这些报文并且回送报文进行响应。

1. 客户服务器进程

假设一个进程为客户，一个进程为服务器。在一个Web中，浏览器是一个客户进程，二web服务器是服务进程；对于一个P2P传输文件，请求文件的对等方是客户进程，发送文件的对等方为服务进程。

所以对客户服务器进程**定义**如下：

在一对进程之间的通信会话场景中，发起通信的进程为客户，在会话开始前等待的是服务器。

1. 进程和计算机网络之间的接口

两个进程通信时必须通过下面的网络，进程通过**套接字（socket）**这个软件接口向网络发送报文和接受报文。**套接字也称之为API**(Application programming Interface), 即引用程序和网络之间的应用程序编程接口。因为套接字存在于应用层和运输层之间，开发者可以控制套接字在应用层端的一切，但是对其运输层部分几乎无法控制，能够控制的权限仅仅有：1. 选择运输层协议（如TCP或者UDP）。2. 设定部分运输层的参数，比如最大报文段长度和缓存大小等。

1. 进程寻址

主机进程需要向另一台主机的进程发送分组时，接受的主机需要有一个地址。为了标识接受进程，有两个信息需要定义：主机地址；目的主机中指定接收进程的标识符。

主机由IP地址(IP address)来标识；而为了标识接收进程，需要知道目的地**端口号（Port Number）**

### 可供应用程序选择的运输服务

有四个方向对应用服务进行分类：可靠数据传输，吞吐量，定时和安全性。

1. 可靠数据传输（reliable data transfer）

运输层协议能够潜在的向应用层程序提供一个进程到进程的可靠服务。发送进程将数据传入套接字就能够相信数据能够无差错的到达接收进程。

1. 吞吐量

吞吐量在这里指的是发送进程向接受进程交付比特的速率。由于其他的session会话要共享本信道带宽，可用吞吐量可能会随着时间波动。具有吞吐量要求的应用成为**带宽敏感的应用**（bandwidth- sensitive application）。而**弹性应用**（elastic application）能够根据当前信道可用的带宽来调整，或多或少的使用吞吐量。

1. 定时

运输层协议需要保证定时。有些服务为了有效性需要而对数据交付有着严格的时间限制；对于非实时应用，较低的时延总比高时延好，但对端到端的时延没有严格的限制。

1. 安全性

运输协议能够为应用程序提供一种或者多种的安全服务。

如数据在发送进程加密，在接受进程解密。除了加密还包括了数据完整性和端点鉴别等。

### 因特网提供的运输服务（TCP、UDP）

TCP：

是一种**面向连接**的服务：在应用层数据流动开始前，TCP需要让服务器和客户主机进行相互交换运输层信息，也就是通过握手提醒客户和服务器大量即将到来的分组。完成三次握手建立后**TCP连接建立在了两个进程（发送和接收）的套接字上**，TCP是**全双工连接**，双方进程都可同时进行分组的收发。报文发送结束时，**TCP连接必须被释放**。

同时TCP是可靠的数据服务：当应用程序把字节流传入套接字，它能够依靠TCP将相同的字节流交付给对方的套接字，没有字节的丢失和冗余。

TCP还有拥塞控制的特性，各个TCP公平的分享网络带宽。

**SSL的引入**：

但是在安全性上需要注意的是：无论是TCP还是UDP都没有加密机制，如果明文通过套接字传入，在网络中传输，明文可能会被嗅探或发现。所以因特网界研制出来**加强版TCP，称之为安全套接字层（Secure sockets layer，SSL）**。 除了完成TCP的功能，SSL提供加密，数据完整性和端点鉴别。SSL不是TCP或者UDP之外的第三种传输协议，**而是通过应用层端对TCP进行的强化**。在客户端和服务端引入SSL（SSL有自己的API，）然后发送进程会向SSL传递明文，发送主机中的SSL把明文加密并且把加密数据传输给网络，接收端从网络得到数据后，SSL对数据进行解密，然后SSL通过它的套接字把明文传输给接收端应用层。

UDP：

UDP是一种不提供不必要服务的轻量级运输协议，只提供最小服务。

UDP无连接并且提供的是不可靠服务。进程把一个报文发送给UDP套接字时，UDP无法保证到达，也不能保证数据到达时不是乱序的。

## 应用层协议

应用层协议定义了以下几个内容：

交换的报文类型；各报文类型的语法；字段语义；一个进程何时发送报文以及如何对报文响应。

### Web和HTTP

#### HTTP概述：

Web网页的应用层协议为**HTTP(HyperText Transfer Protocol)超文本传输协议**.该协议定义了在客户程序和服务器程序交换的报文结构以及交换的方式。

接着对Web介绍。**Web page页面**是由对象组成，一个**对象(object)**指的是一个文件，比如一个HTML文件，JPEG图片，java小程序或者视频片段。这些文件可以通过URL寻址。多数Web包含一个**HTML基本文件（base HTML file）**和几个引用对象。因为**WEB浏览器（Web browser）**实现了HTTP客户端，Web服务器（Web server）实现了服务端，其用于存储Web对象。流行的服务器有Apache和Microsoft Internet Information server微软互联网信息服务器。

HTTP定义了浏览器向Web服务器请求页面的方式，以及服务器向浏览器（客户）传送web page。当用户请求一个web页面时，浏览器向服务器发出该页面中包含对象的HTTP请求报文，服务器接收到请求，并且用这些报文进行响应。

**HTTP使用TCP作为运输协议：实现可靠数据传输，不关注下层运输状态，只管通过套接字收取和发送报文。**

**HTTP是一个无状态协议（stateless protocol）：**服务器并不会储存任何客户状态信息。

**HTTP默认端口**：80

#### 持续连接和非持续连接

当客户在一个长时间范围内通信时，客户可以发出一系列的请求并且服务器需要对每个请求进行响应。这一系列的请求可以是周期性的或者不间断的，那么此时有所有的求和响应是通过一个相同的TCP连接（**持续连接persistent connection**），还是其实每一个请求响应是由不同的独立的一个TCP连接完成(**非持续连接non-persistent connection**)？

**HTTP默认情况下是持续连接**的：所有请求响应由同一个TCP连接完成。

但是HTTP连接情况是可以被配置为非持续连接的。

1. **采用非持续连接的HTTP：**

在非持续连接的HTTP上请求一个Web页面的步骤：

1. HTTP客户进程向端口号80发起一个服务器（如[www.shareBike.com](http://www.shareBike.com)）的 TCP连接请求
2. HTTP客户经过套接字向服务器发送一个请求报文，这个HTTP请求报文里包含了路径名如/customer/home.index
3. HTTP服务器进程通过套接字接收报文，从其存储器中检索出对象[www.shareBike.com/customer/home.index](http://www.shareBike.com/customer/home.index)，在一个响应报文中封装对象，通过套接字发送这个报文。
4. 当TCP确认客户完整收到报文后，HTTP服务器进程通知TCP断开连接
5. 客户此时已经收到了响应报文，从响应报文提取HTML和这个网页上对象的引用，并且此时已经断开TCP连接。
6. 对每一个对象应用都会重复前几个步骤

当浏览器收到响应报文后会向客户显示该页面，不同的浏览器可能解释该页面的方式不同（比如使用IE和谷歌），而HTTP只定义了HTTP的客户和服务器之间的通信协议。

**所以对于采用非持续连接，每一个对象的传输都有一次tcp连接的建立和释放，当一个页面有一个html基本文件和10个jpeg图片时，这个页面需要11次tcp的建立来传输。**

默认情况下，大部分浏览器会同时进行5-10个并行的TCP连接，每一条连接处理一个事务。如果此时把最大连接数改为1，则形成了串形连接。但是并行连接可以缩短响应时间。

请求并且接收到服务端文件的时间消耗为2个RTT（往返时延）+HTML文件传输时间

因为三次握手的前两次占据了一个RTT，第三次握手时客户端向服务端发送确认报文，第三次握手时可以传输数据，所以会携带请求报文内容；服务端一旦收到请求报文，在TCP连接上发送HTML文件。所以第二次的HTTP请求和响应需要一个RTT，加上HTML的传输时间。

1. **采用持续连接的HTTP**

采用非持续连接的HTTP有两个缺点：每一个请求对象都需要建立和维护一个全新的TCP连接，客户和服务器都需要分配TCP缓冲区和保持TCP变量，对于Web服务器来说这是很大的开销；同时交付一个对象需要2RTT+HTML对象传输时间

采用持续连接的HTTP：当一条连接在经过一个配置的超时间隔内仍未被使用，此时HTTP服务器断开该连接。当连接没有断开时，在相同的客户和服务器之间可以传输多个对象，是的一个完整的页面能够通过一个TCP连接完成传输。甚至，多个服务器对同一个客户的传输时，可以采用单个持续的TCP连接，对象的请求可以一个个的发出而不必等待未决定请求（流水线）的回答

#### HTTP报文格式

1. HTTP请求报文

文本

描述已自动生成

如上图，该HTTP报文由5行组成。HTTP请求报文的**第一行为请求行（request line）**，后继的行都叫做**首部行（header line）**

**请求行组成**由三个字段：**方法字段**（Get, Post, Pull, Delete）；**URL字段**（带有请求对象的标识，如上图的page.html）；**HTTP版本字段**

首部行：

1. Host表示了主机，这是Web高速代理缓存所需要的。
2. Connection: close， 表明了不要麻烦的使用持续连接，服务器发送请求对象后就关闭TCP
3. User-agent:在首部行中表明了用户代理，即向服务器发送请求的浏览器类型。Mozilla/5.0就是firefox浏览器。
4. Accpet-language:fr表示用户想要得到的是法语版本，否则发送默认版本。
5. HTTP响应报文：

文本, 信件

描述已自动生成

响应报文有三个部分：一个**初始状态行（status line）**，6个**首部行（header line）**，**实体体（entity-body）**。实体体是主要部分，包含了请求的对象本身。

**状态行有三个字段：协议版本，状态码，状态信息**

200 OK：请求成功

301 Moved Permanently: 重定向，请求的对象已经被转移了，将自动获取新的URL。

400 Bad request：通用差错代码，请求无法被服务器理解。

404 Not found：请求对象不在服务器上。

500 一般是服务器报错，如

505 HTTP version not supported：HTTP版本协议不被服务器支持

首部行：

Connection: close，告诉客户发送完报文后即将关闭TCP连接

Date：服务器响应时的报文日期和时间

Server：类似于请求报文中的user-agent，这里表明服务是由一台Apache Web服务器产生

Last-modified: 该对象上一次创建或者修改的时间，它对本地客户在网络缓存起（代理服务器）上的对象缓存非常重要。

Content-length: 被发送对象的字节数

Content-type: 表明被发送对象类型。

1. 一个通用的HTTP报文：

图示

低可信度描述已自动生成

Sp为space，crlf 回车换行符

当然这里除了请求行和首部行，还携带了一个实体体(entity body)

当使用Get方法时，实体体为空

用户提交表单时使用Post方法，实体体中包含的就是用户在表单字段中输入的值。

Head方法类似于Get方法，服务器收到Head时返回响应报文，但是不返回请求对象，所以Head常用于开发者的调试和跟踪。

Put允许用户上传特定的对象到Web服务器的指定路径。

Delete允许用户删除Web服务器上的对象。

#### Cookie

由于HTTP是无状态协议，但是有时一个Web站点需要识别用户，可能是因为限制访问或者联系内容和用户身份（如用户偏好），为此需要使用到cookie。

Cookie有四个组件：

1. 在HTTP的响应报文中有cookie首部行
2. 在HTTP的请求报文中有cookie首部行
3. 在用户端系统中保留了一个cookie文件，并且由浏览器进行管理
4. 位于Web站点的后端数据库

使用cookie的过程：

图示

描述已自动生成

如图左边用户端系统保留了一个cookie文件，用于存储用户访问服务器留下的cookie，可以看到一开始用户访问过eBay这个网站，并且已经设置了cookie识别码为8734

此时，用户访问亚马逊Amazon网站，当用户的请求报文到达了amazon服务器，该web服务器此时产生一个cookie的唯一识别码，并且以此为索引在其后端数据库产生一个表项。接下来，amazon服务器在给客户的响应报文中，需要在首部行加上Set-cookie，例子中的识别码为1678. 用户的浏览器接收到了来自于服务器的响应报文，并且在首部行中读取到Set-cookie。浏览器在它管理的cookie文件中加上这一行。

在这种情况下，用户接着浏览amazon的网站，每请求一个页面，用户浏览器查询cookie文件并且抽取cookie识别码，放到HTTP请求报文中的首部行。所以Amazon服务器可以跟踪用户站点的活动，比如Amazon使用cookie来支撑购物车服务：维护用户加入购物车的列表，在最后一起结算。

当然如果一开始在表单用户填写了自己的个人信息，这些信息也会随着cookie被传输。但是大多数时候cookie都是用作**用户标识**：浏览器向服务器传输cookie首部，服务器所以认识了用户。**因此，cookie可以在无状态的HTTP协议上添加一层会话层**。

#### Web缓存（代理服务器）

Web cache缓存==proxy server代理服务器，初始服务器来满足HTTP请求的网络实体。

Web缓存由自己的磁盘储存空间，并且在储存空间中保留最近请求过对象的副本。

图示, 示意图

描述已自动生成

Web缓存的工作过程：

假设浏览器正在请求<http://www.someschool.edu/campus.gif>对象

1. 浏览器首先创建一个Web缓存器的TCP连接，并且向Web缓存器的对象发送一个HTTP请求
2. Web缓存器检查本地是否储存了该副本，如果有则向客户浏览器发送HTTP响应报文，返回该对象
3. 如果没有副本，则打开一个该对象的初始服务器[www.somechool.edu的TCP](http://www.somechool.edu的TCP)连接，Web缓存器发出HTTP请求该对象，服务器接收到请求发回响应报文。Web缓存器收到报文后，在本地储存副本，并且向客户浏览器发送HTTP响应报文来发送副本。

所以代理服务器（Web缓存）即使客户也是服务器，它大大降低了因特网上的Web流量，改善所有应用的性能。