

## 第二章 应用层

本层原理、流行应用原理和相关协议、TCP 套接字（Socket）编程、UDP 套接字编程

目标：

1. 网络应用的**原理**：网络应用协议的概念和实现。服务模型、两种服务模式、内容分发。
2. 网络应用的**实例**：流行的应用层协议。
3. **编程**-Socket API：原语。

互联网中应用层协议最多：可以自定义功能所以协议特别多。很快地设计和部署新的业务模式，鼓励创新。存在各种各样的网络应用。

创建一个新的网络应用：编程、租服务器、装环境、搭服务器、提供客户端下载。**网络核心中没有应用层软件**，仅在端系统部署，非常块。

### 2.1 应用层协议原理

#### 体系结构

1. **客户-服务器**模式：（C/S:client/server）

服务器一直运行、IP固定；客户端可以下线、IP可变。一般仅客户端主动与服务器相互通信，间歇性连接，客户端之间不相互通信。——可扩展性差、设备增加性能断崖式下降、可靠性差：服务器坏掉直接无法服务。

2. **对等**模式：（P2P:Peer To Peer）

可以平滑扩展，但是服务能力不稳定、管理困难。

3. **混合体**模式：客户-服务器和对等体系结构。

Naspter（客户端有音乐文件，上线后向服务器发送消息，服务器告诉请求者从哪里下载文件，用户越多下载速度越快——查询是集中式、文件分发是p2p）。

集中通信 QQ 等，在线检测：集中；用户通信：P2P。

#### 进程通信

进程：在主机上运行的应用程序。——同一主机使用操作系统管道即可。

客户端进程、服务器进程。——P2P架构每个会话也有这两种分别。

需要解决的问题：

1. 进程**标示和寻址**。
2. 传输层向应用层提供**服务形式**（接口）。位置：层间界面的SAP 形式：应用程序接口API。
3. 定义**协议**，如何实现应用功能。

**问题1**：标示和寻址

主机IP、传输层协议（TCP or UDP）、端口号（进程跑在哪里）。

一个进程：用**IP+port**标示 端节点，一对主机进程之间的通信由**2个端节点**构成。

**问题2**：服务形式

- 穿过的信息：

要传输的报文（本层SDU）、谁传的、传给谁。

传输层实体封装的内容：源端口号，目标端口号，数据等。将IP地址往下交IP实体，用于封装IP数据报。

- 减少信息量（代表层间信息）——**socket：一个整数**

TCP代表四元组（源、目的IP端口）UDP代表二元组（本地的IP和端口）。

socket是**本地**为了便于表示使用的，不是公用的。穿过层间信息少、便于管理。

socket和端口的区别！——其实就是上一章讲的ICI。

- **TCP-socket**

4元组：(源IP, 源port, 目标IP, 目标port)

唯一的指定了一个会话

- **UDP-socket**

但是传输报文时：必须要提供**对方IP, port**。接收报文时：传输层需要上传对方的IP, port。

也就是这两部分不在socket表示，有单独的数据结构来表示。

指定了应用所在的一个端节点（end point）

进程向套接字发送报文或从套接字接收报文——类比于门。

**问题3：**如何实现（编制协议）

**应用层协议：**报文格式，解释，时序等。可以进行传输、解析、实现时序。

应用协议仅仅是应用的一个**组成部分**。实体仅是涉及到层间通信的处理部分（运行中的软硬件模块），其他业务部分不是实体。

常见协议：HTTP, SMTP（公用）、Skype（私用）。

## 其他知识点

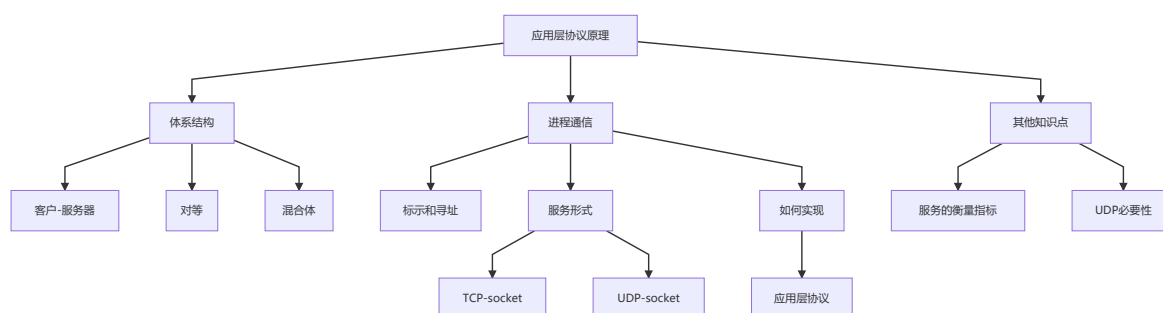
传输层服务的**衡量指标**：数据丢失率、延迟、吞吐、安全性。——常见应用对传输服务的要求

传输层提供的服务：TCP（可靠、流量控制、拥塞控制、面向连接；时间保证、最小吞吐保证和安全无法提供）、UDP（不可靠，没有各种保证、没有链接建立过程）。

**UDP存在的必要性：**

区分不同的进程、无需建立连接、不做可靠性的工作、能够按照设定的速度发送数据。

安全性：TCP & UDP**都不安全**——明文传输。可以使用SSL，在TCP和应用之间，属于应用层的功能。私密性、完整性鉴别和可认证性。



## 2.2 Web and HTTP

## 一些术语

**web**: 一种**应用**，由对象组成——支持**协议**: **HTTP**。

**对象**: 各种网页中的资源（使用url来唯一表示）。

基本的HTML文件+引用连接（url）。各种对象通过相互引用构成了网状的信息空间。

**url**: 统一资源定位系统。

```
Prot://user:psw@www.someSchool.edu/someDept/pic.gif:port
```

协议名 :// 用户:口令 @ 主机名 / 路径名 :端口

注: url支持匿名访问, 所以用户: 口令可以省略。http默认80端口, ftp默认21端口, 所以url可以不指定。

## HTTP概况

HTTP: **超文本传输协议**。应用层Web服务的实现协议。

**客户/服务器**模式: 客户端发送request, 服务器响应response。

两类: 1.0:RFC 1945/ 1.1:RFC 2068

跑在**TCP**协议之上:

1. 客户发起与服务器的 TCP连接 (建立套接字)。
2. 服务器本身有一个守候 (waiting) socket, 客户端发送请求后就建立另一个通信socket。
3. 客户端拿到响应内容之后在本地渲染网页。
4. 完成以后关闭连接。

HTTP是**无状态**的: 服务器并不维护关于客户的任何信息。

——支持更多用户、避免死机同步、避免维护历史信息。

## HTTP连接

需要**下层实体进行一次交互**才能完成。

区别:

- 非持久HTTP: 1.0版本, **最多只有一个**对象在 TCP连接上发送。
- 持久HTTP: 1.1版本, **多个对象**可以在一个TCP连接上发送。

**非持久**:

- 每次都要重复连接建立、接收、请求、响应。
- 往返时间RTT (分组从客户端到服务器再回来-忽略小分组传输时间)。
- 总共2RTT+传输时间。

**持久**:

- **非流水线** (non-pipeline) : 收到前一个响应后 才能发出新的请求, 收到一个对象花费一个RTT。
- **流水线**: 一个引用对象就立即产生一个请求, 甚至可以达到仅用一个RTT。

好处: 可以省掉每次建立连接的一个RTT, 避免并行建立TCP连接占用资源。

## HTTP请求报文

分类: **请求**、**响应**。ASCII码编码, 人类可读。

**组成：**

1. 请求行（GET-请求/POST-上载/HEAD-拿头，文件路径，协议版本号）。
2. 首部行/headline（key:value格式：主机名、浏览器版本、可以设置非持久连接）。
3. 后接回车空行。
4. 实体行（请求报文通常没有）。

**提交表单输入：**

1. Post方式：输入内容放在实体行
2. url方式：在url后边 ?key=value&key2=value2

**方法类型：**

- HTTP/1.0: GET POST HEAD
- HTTP/1.1: GET POST HEAD; PUT DELETE (网页管理员使用)

## HTTP响应报文

**组成：**

1. 状态行（协议版本号，状态码和解释）。
2. 首部行（key:value格式：连接方式、服务器版本、上次修改日期-版本号、内容长度-tcp不维护界限、内容类型）。
3. 回车空行。
4. 实体行。

**响应状态码：**

200 OK; 301 Moved Permanently; 400 Bad Request; 404 Not Found; 505 HTTP Version Not Supported

可以在自己电脑上测试手动发送客户端请求。

## cookies

**含义：**用户-服务器状态——4个组成部分

1. HTTP响应报文有一个cookie的首部行-服务器状态;
2. HTTP请求报文有一个cookie的首部行-客户端状态;
3. 用户端系统存储cookie文件，用于浏览器管理;
4. 在Web站点维护一个后端数据库存各客户cookie。

例子，初始请求无cookie，维护cookie-ID，以后请求就带了这个cookie-ID。

**作用：**

1. 用户验证
2. 购物车
3. 推荐
4. 用户状态

**问题：**隐私问题。

## Web缓存-cache

**目标：**不访问原始服务器，就满足客户的请求。

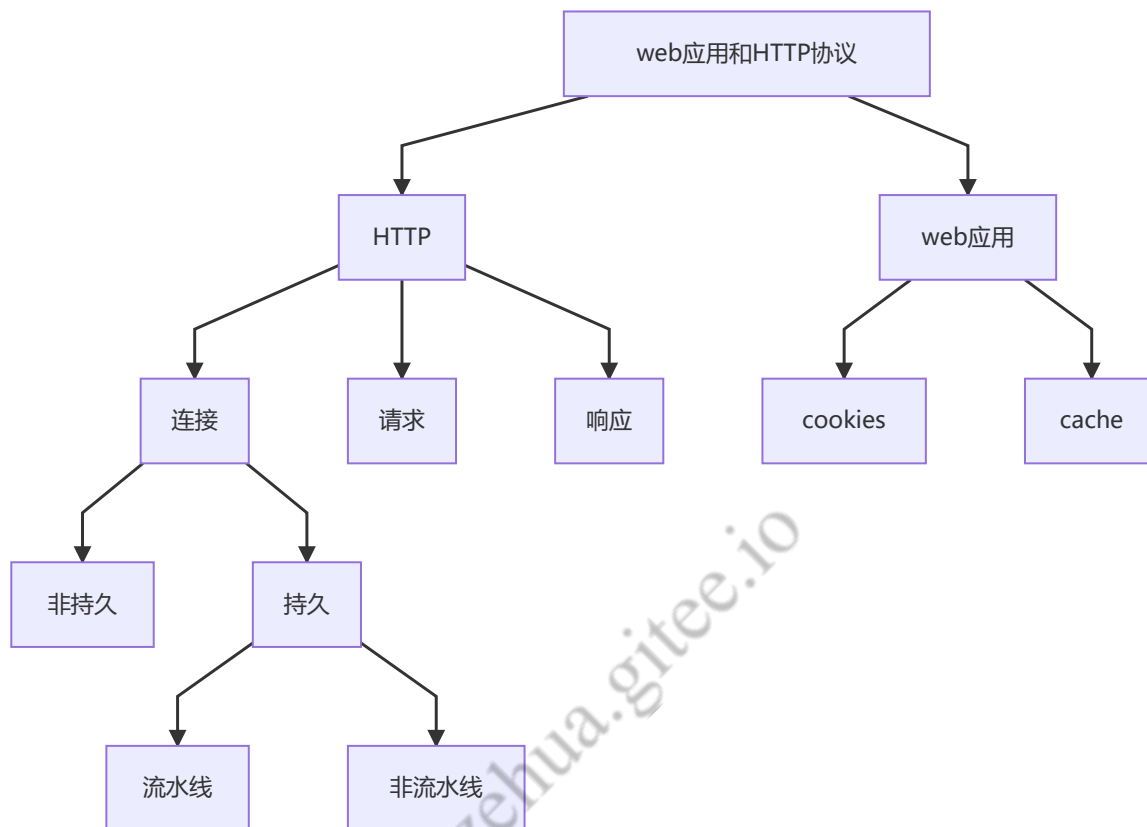
代理服务器将内容缓存下来，客户之间请求代理服务器。命中直接返回——**客户端快，服务器端负载轻，网络负担也减弱**。未命中继续请求原始服务器。

通常缓存是由ISP安装，既是客户端又是服务器。互联网二八定律——20%热点内容80%人访问，使得缓存成为可能。

示例：没有缓存，排队延时接近无限大。两种方式

1. 增大带宽，更大的接入互联网速度。
2. 安装本地缓存，在40%命中率的条件下平均响应时间甚至还减小了。

**问题：**远程服务器内容变了。解决方法：**条件GET方法。If-modified-since**



## 2.3 FTP\*

文件传输协议（了解）

客户/服务器模式（还有管理系统、用户界面等等），端口号为21，可以upload也可以download。

可以建立连接，身份认证（口令都是明文传输），然后可以发送命令-上载下载list等等，服务器单独使用20号端口向客户端发送TCP数据连接。控制连接：**带外**（“out of band”）传送。区别于http的“带内”。

FTP天然就是**有状态**的协议，而http无状态，只能靠cookie打补丁。

一些控制命令和响应，都是ASCII码明文传输。

## 2.4 EMail

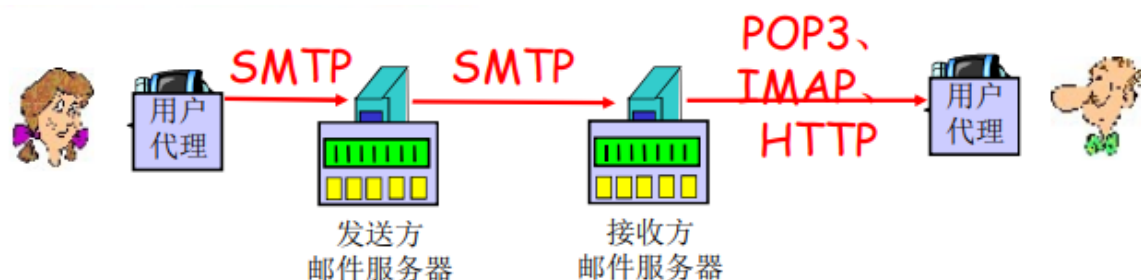
### 概述

3个主要组成部分：

- **用户代理**（客户端软件：浏览器、outlook软件等）
- **邮件服务器**

□ 简单邮件**传输协议**：SMTP

用户代理：又名“邮件阅读器”。



**过程**：用户代理发给邮件服务器，邮件服务器放到队列当中，使用SMTP将其发到对应其他邮件服务器。目标服务器收到自己的用户邮箱当中，运行客户代理从自己的邮箱中拉取内容（POP3等协议）。

## SMTP

服务器守候在25端口，接收端作为服务器，发送方作为客户端。

三个阶段：□握手 □传输报文 □关闭

命令/响应报文：**ASCII码**，包含**状态码**和**状态解释**。

**报文必须为7位ASCII码**

一个发邮件的例子。为什么要放到队列里呢？1. 邮件很多要排队处理、2. 定期发送可能不是实时都有邮件。

伪造用户发邮件很容易，全部是**明文传输**。可以传英文，中文就不行了，文件也不能传。

建立连接后可以发很多的邮件，不用每次都重新建立连接。——**持久连接**

可以通过telnet命令自己尝试和邮件服务器进行交互。

**总结**：持久连接、7位ASCII编码、SMTP服务器使用 CRLF.CRLF结尾。

与HTTP比较：

- 相同：都是ASCII形式的命令
- 区别：HTTP拉，SMTP推；HTTP一次响应报文至多一个对象，SMTP同时把所有对象打包；HTTP有非持久连接模式。

## 报文格式

文本**报文的标准**：

- 首部行：To: From: Subje: (主题) CC: (抄送)、密送等等。
- 主体：ASCII码字符。

**多媒体扩展**（MIME）：base64编码

把若干个不在ASCII范围的内容编码到ASCII，同时支持解码。

## 访问协议

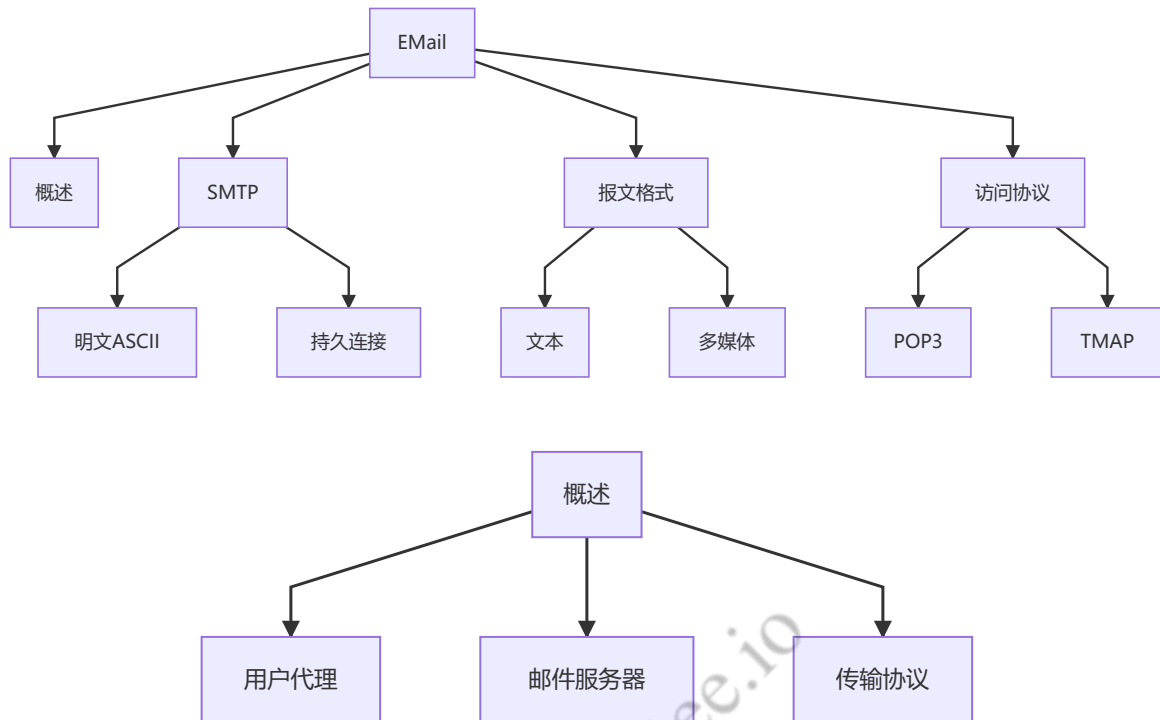
前两跳是推，最后一跳是拉。包含POP3、TMAP、HTTP。

- POP3：邮局访问协议，包含身份确认。
- IMAP：在POP3基础上包含远程目录维护等功能。
- HTTP：web、文件上下载、收邮件。

POP3:

1. 用户确认阶段：用户名密码。
2. 事务处理阶段：通过命令进行操作。（下载并删除、下载并保留）会话**无状态**。

IMAP：保留用户状态，支持远程目录维护。



## 2.5 DNS

### 概述

**功能：域名到IP地址的转换。**

DNS不是给人直接用的，是给其他应用用的（web、FTP）——基础设施

必要性：IP地址用于标识、寻址，很难记。用户使用主机的域名，字符串转换为IP让网络进行寻址。

要解决的问题：

1. 如何**命名**：层次化命名。
2. 如何完成**转换**：分布式维护。
3. 如何**维护**：增加删除的方法。

历史：

- ARPANET：没有层次、同时在一个平面；一台主机提供解析服务。——重名太多、很难管理。

主要思路：

- **分层命名**。
- **分布式数据库**。
- **UDP 53端口**，不需要握手。
- 互联网的**核心功能**——反而是靠边缘设备来实现的。

主要目的：

- 主机名-IP地址的转换。name/IP translate——**核心功能**
- 主机别名（管理）到规范名字（访问）的转换。Host aliasing

- 邮件服务器别名和正规名字转换。Mail server aliasing
- 负载均衡。Load Distribution

## 名字空间

(问题一)

**结构：**层次树状结构——几百个顶级域、若干个二级域、许许多多子域，树叶是主机。

顶级域：**通用、国家**。 .com国际公司、.edu学校；.cn中国、.us美国。域名越靠后域越高级，对主机做标识，从树叶往树根走。对域做标识，从树枝往树根走。

有13个根名字服务器——可靠、便于维护（每次从树根找总能找到）

**域名：**从本域往上，直到树根；使用“.”间隔；可以有主机的主机和域的域名。

**域名的管理：**

子域的命名由主域定义，不同主域下子域命名完全有可能不同。

域与物理网络无关：（域的划分是逻辑的）

- 一个域的主机可以不在一个网络
- 一个网络的主机不一定在一个域

## 名字服务器

(问题二)

一个名字服务器的问题：可靠性、扩展性、维护。

**区域(zone)：**

- 将DNS名字空间划分为互不相交的区域。
- 每个域有一个**权威名字服务器**（可以在区域外）。

一个分区的例子。上层权威域名服务器维护一个到下层权威域名服务器的指针。

TLD服务器：顶级域服务器。

区域名字服务器维护**资源记录**：(resource records)

1. 子域情况
2. 域名-IP地址（别名等）映射情况

**资源记录格式**（数据库设计）：域名-Domain\_name、生存时间-Ttl（缓存记录为有限值-默认两天；权威记录为无限）、类别-Class（都是IN）、值-Value（IP地址或其他）、类型-Type（存储的资源的对应类型）。

**Type字段：**

- Type=A：主机+IP
- Type=CNAME：别名+规范名字
- Type=NS：子域的域名+子域权威服务器的域名
- Type=MX：邮件服务器别名+正规名字

对于主域服务器内要找子域必须要有两条记录（服务器是谁-域名、服务器在哪-IP）

**DNS工作过程：**

1. 应用调用解析器。
2. 解析器访问**Local Name Server**。（自动配或者手动配，一般在一个子网）
3. Name Server返回响应。



上网所需四个参数：IP地址、子网掩码、default gateway（默认网关）、DNS local Name Server。

本地名字服务器（Local Name Server）：并不严格属于层次结构。

名字服务器(Name Server)**名字解析过程**：

- 在Local Name Server（域名在子网/有缓存）：**直接返回**。
- **递归查询**：根服务器负担太重。
- **迭代查询**：根（及各级域名）服务器返回是下一个NS的地址。学到以后可以缓存下来。

## 报文格式

查询和响应报文**格式相同**（不用硬记）

- **标识符**（ID）：16位——重要：支持流水线的方式同时查询多个。
- 标志位（flags）
- 其他字段（略）

**缓存**：学到就缓存，TTL（默认2天）

## 维护问题

（问题三）

在上级域的名字服务器中增加两条记录，指向**子域的域名**（名字服务器的名字-typeNS）和**域名服务器的地址**（typeA）。

子域名字服务器自行维护自己下面的域名。

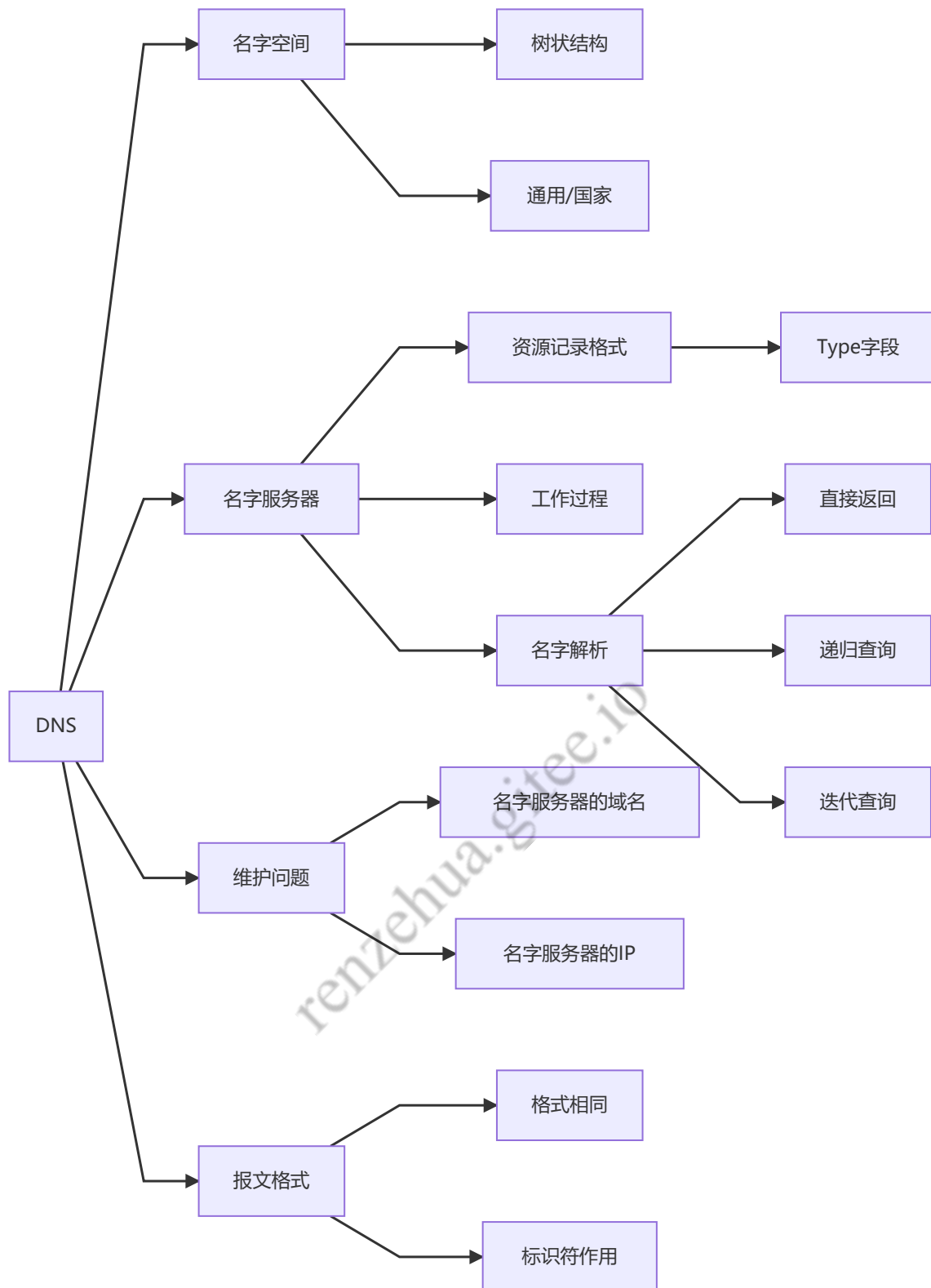
## 安全问题

——相对来讲还比较可靠。

DDoS 攻击：对根服务器、对TLD服务器——效果一般（有缓存）

重定向攻击：中间人攻击、DNS中毒——技术上较困难

利用DNS基础设施进行DDoS：伪造IP、查询放大——效果有限



## 2.6 P2P应用

一类应用。——C/S模式可靠性可扩展性的问题。

请求节点增加、提供服务节点也在增加。没有（或极少）一直运行的 服务器。BT-文件分发、KanKan-流媒体、Skype-VoIP（网络电话）

### 与C/S对比

例文件分发：

- 一台服务器分发文件（大小F）到N个peer

$$D_{c-s} \geq \max\{NF/u_s, F/d_{\min}\}$$

$d_{\min}$ -客户端最小下载带宽。在客户端量很大的时候，瓶颈在于服务器。随着客户端增加下限线性提升。

- 最少需要上载一份拷贝F，所有客户端总体下载量NF

其他所有的peer节点都可以上载

$$D_{P2P} \geq \max\{NF/(u_s + \sum u_i), F/d_{\min}, F/u_s\}$$

客户端增加性能下限变化缓慢。

P2P模式的问题：可管理性不好、动态变化资源可能找不到。

## 几种管理方式

ppt上在后边

overlay（覆盖网）Peer节点构成的提供服务的抽象网络。

在存储文件时，有文件源数据、文件描述、哈希值（唯一确定用于查找）

### 1. 非结构化：overlay是随机的、任意的 以文件共享系统为例

文件共享面临的问题：如何定位资源、如何处理上下线

- **集中化目录：Napster** 资源管理中心化，文件分发存储分布式——单点故障、性能瓶颈、侵犯版权。
- **完全分布式：Gnutella** 没有中心服务器，开源。用泛洪（flooding）的方式发送查询，向所有邻居进行查询，邻居再向邻居发送查询。安装好后有一个常在线设备列表，发送ping，转发ping，都回复pong，随机建立连接。退出时和邻居节点打招呼，邻居再连一条保持连接度——查询请求过多（限制范围洪泛查询）、效果不好。
- **混合体：KaZaA** 分组员组长。组长之间的关系类似于Gnutella全分布式、组长与组员之间类似Napster中心与边缘。使用描述符匹配，返回元数据、哈希值和IP。

处理技巧：请求排队、激励优先权、并行下载

典型例子：BT（**BitTorrent**）——一个节点加入到一个**洪流**（Torrent）中，可以共享资源。把资源划分为很多小块，使用**bitmap**标识当前设备有没有这一部分资源。所有节点在洪流中泛洪bitmap，可以知道哪些节点拥有那些资源块。

**请求块**：刚加入时bitmap全是0，请求四个块（稀缺优先）。周期询问、按需请求。

**发送块**：服务越多性能越好——报还一报tit-for-tat（个人利益和集体利益相捆绑）。同时只服务4个最好的（有限疏通）；每隔30秒，两个周期新选择带宽大的，第三个周期随机选一个（**优化疏通**）可以发现更好的伙伴（启发算法思想）。

**扰动**churn: peer节点可能会上线或者下线。

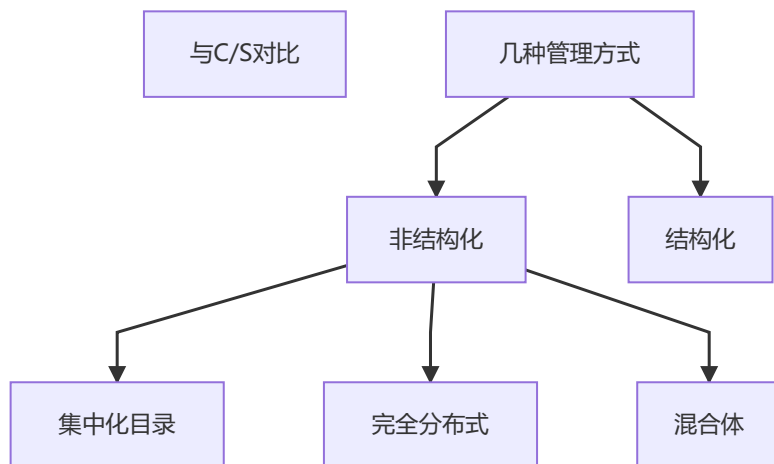
节点拥有文件的全部（种子）、都没有（吸血鬼）。拥有全部可以选择离开或者留下。

加入洪流的过程是带外进行的。在维护网站检索资源列表，点击torrent文件，包含了tracking server（跟踪服务器），tracking server维护当前上载下载的过程，再获取peer列表，建立连接进行上下载。

### 2. 结构化（DHT）：overlay有规律（环、树等等）

哈希表、DHT方案、环形DHT、Peer波动

例如，维护一个环状网络，节点和内容id是重叠的，节点哈希值对应内容哈希值空间——知道到哪去查。



## 2.7 CDN

内容分发网络。

web访问加速功能服务的内容分发系统——**分布式应用层基础设施**

网络中**杀手级业务**（占用大量带宽的业务：视频等等）

问题：**规模性**——如何同时提供大量传输、**异构性**——设备异构要求各不相同。

### 多媒体流化服务

视频可以认为是一系列图像的序列，图像是一系列像素点的序列，视频同时还包含音频。在传输的时候要经过压缩——**时间空间冗余度**。

压缩方式：CBR——固定码率；VBR——码率随时间动态变化。

例如：AVS、MPEG 1、MPEG 2、MPEG 4。可伸缩编码。

**流化服务**streaming：“边下边看”，让用户的点播延迟大大降低。

常见流化服务协议：**DASH**——Dynamic, Adaptive Streaming over HTTP（在HTTP上动态自适应流化播放）

- 服务器端：
  1. 将视频文件分割成多个**块**。
  2. 每个块独立存储，有不同码率。
  3. **告示文件**（manifest file）包含了每一块的url、码率、解析度等信息。
- 客户端：
  1. 下载告示文件（不同块可能在不同服务器中）。
  2. 周期性测量带宽，根据需求、缓冲区和延迟情况动态调节向哪里请求。
  3. 一个时刻请求一个块。——带宽够，请求最大码率，不同时刻可以切换请求源。

解决不同客户端需求和多网络服务器情形下的方案。**客户端自适应**决定什么时候、什么码率、请求源。

存在的**问题**：高并发服务器性能瓶颈、链路存在带宽瓶颈、单点故障、重复传输造成流量浪费。——CDN的出现

### CDN

全网部署缓存节点，就近向用户提供服务。ICP买CDN运营商的服务，以更好地向用户提供**内容加速服务**。客户端经过域名解析重定向找到最近CDN节点。（中国蓝汛）

主要**分类**（两种）：

1. enter deep：部署在local ISP，离用户近，非常快但是管理困难。（深入群众）
2. bring home：部署在关键ISP节点附近，部署节点很少，离用户较远。（抢占关键少数位置）

**实现**：（让内容靠近用户）

- ICP在CDN节点中存储内容的多个拷贝
- 用户在告示文件获取块的位置
- 域名解析重定向到最近的拷贝，请求内容

**特点**：over the top

- 在互联网最顶层（应用层）实现
- 在网络边缘提供服务

**OTT挑战**：从哪个节点获取？如何切换、拥塞控制？节点如何部署？

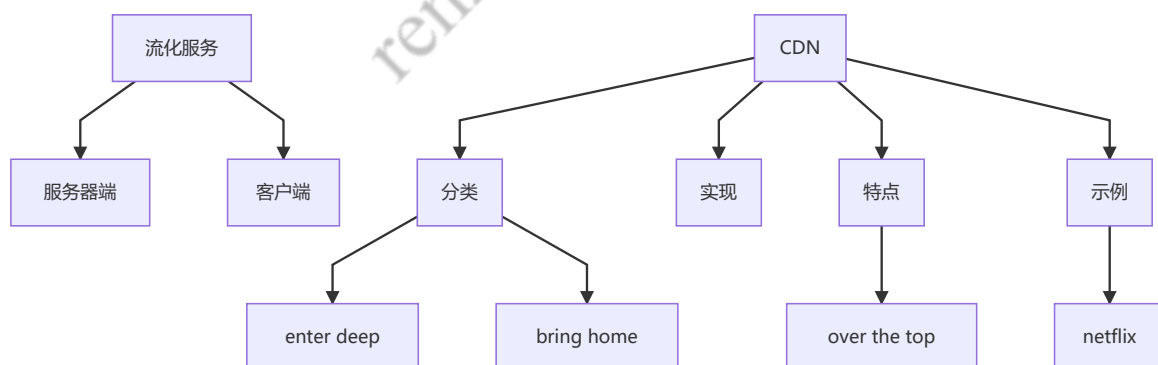
**示例**：

- 小示例：ICP（netcinema）、CDN（kingCDN）。包含主服务器和权威名字服务器。

1. Bob访问ICP获取视频url
2. 向netcinema名字服务器请求返回重定向url
3. 向kingCDN名字服务器请求url
4. 在kingCDN的cache节点请求流化服务

对用户是透明的，用户察觉不到。

- 经典示例：网飞Netflix
1. 包含用户注册管理服务器
  2. 制作后上载到云端（租用服务器），同时租用CDN服务
  3. 每次用户登录以后点击网页域名解析重定向
  4. 由CDN节点向用户提供服务



## 2.8 TCP 套接字编程

### Socket编程总述

传输层向应用层各种协议提供报文传输服务。

位置：Socket；形式：Socket API。

本章讲应用层要怎么调用socket、使用socket、关闭socket来实现传输。

**分类**：

1. **TCP**：可靠的、字节流的服务（保证流，不保证报文界限）。面向连接。

2. **UDP**: 不可靠 (数据UDP数据报) 服务。无连接。

socket: 也是一个**整数** (两个端节点的代表-四元组) 应用进程与端到端传输协议 (TCP或UDP) 之间的**门户**。

TCP服务: 从一个进程向另一个进程**可靠**地传输**字节流**。

socket API是一套函数 (加粗都是函数)

## 概述

1. 服务器先运行

- **创建**一个socket (什么都不是) ——整数
- 和本地端口**捆绑**——welcome socket (本地IP+端口)
- 阻塞式**等待接收**用户的连接 (没来就不执行)

2. 客户端主动连接

- **创建**客户端本地socket (**隐式捆绑**到本地port)
- 根据服务器IP+端口, 与服务器进程**连接**, 未回复就是阻塞

3. 客户端连接请求到来时

- 服务器**接受**请求, 解除阻塞。
- 返回一个新的socket (connection socket)
- 在新socket上收发, 支持同时多个, 使用源IP+port区分
- 传输完成**关闭**connection, welcome仍在

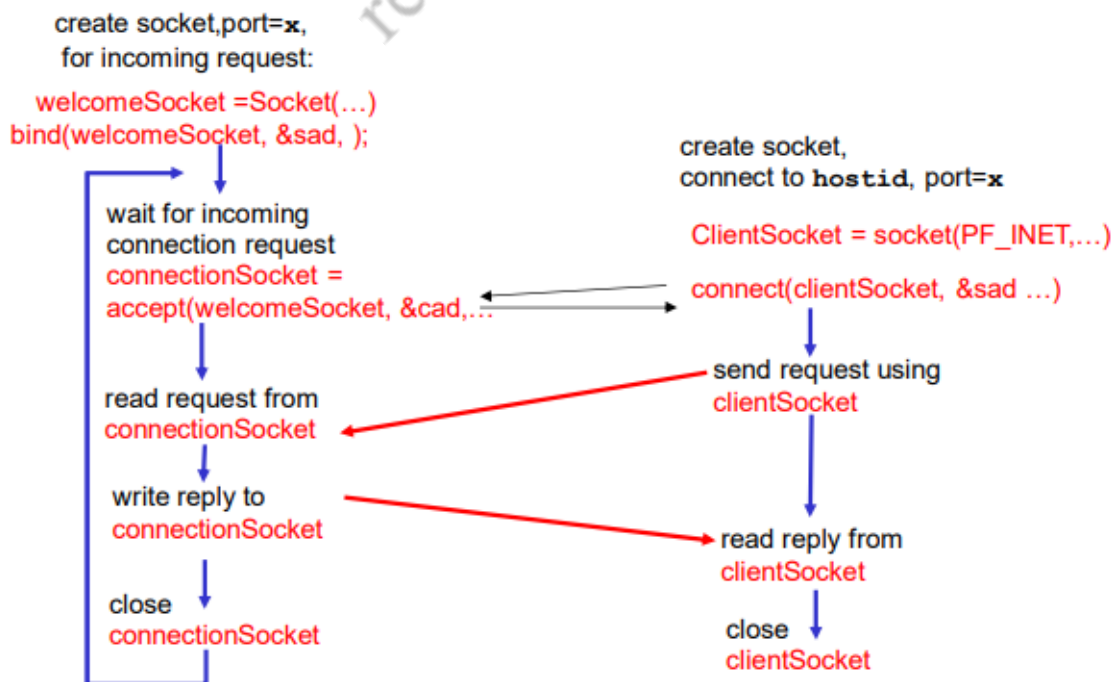
4. 连接API调用有效时, 客户端与服务器建立了TCP连接。

应用程序角度: TCP在客户端和服务进程之间提供了**可靠的、字节流** (管道) 服务

## 应用样例

详细流程:

客户端从键盘读取字符发给服务器、服务器读取、转大写发给客户端、客户端读取并打印。



黑色代表TCP报文交互, 红色代表应用报文交互。

1. welcomeSocket = Socket(...) **创建**欢迎socket ——服务器

2. bind(welcomeSocket, &sad, );和本地端口**捆绑**——服务器
3. connectionSocket = accept(welcomeSocket, &cad,... **等待**连接建立请求，没有请求就阻塞——服务器
4. ClientSocket = socket(PF\_INET,...)**创建**本地socket ——客户端
5. 隐含了bind 和本地某个端口**捆绑**——客户端
6. connect(clientSocket, &sad ...) **连接**请求（应答返回以后解除阻塞）——客户端
7. connectionSocket = accept(welcomeSocket, &cad,... 收到请求返回应答，建立新的connection socket，包含客户端IP端口。——服务器
8. write **发送**需要的资源请求——客户端
9. read **读取**接收请求并解析——服务器
10. write 计算出结果**发送**回复——服务器
11. read **读取**服务器返回值，输出显示——客户端
12. close 关闭连接，删掉connection socket——服务器
13. close 关闭连接，删掉client socket——客户端

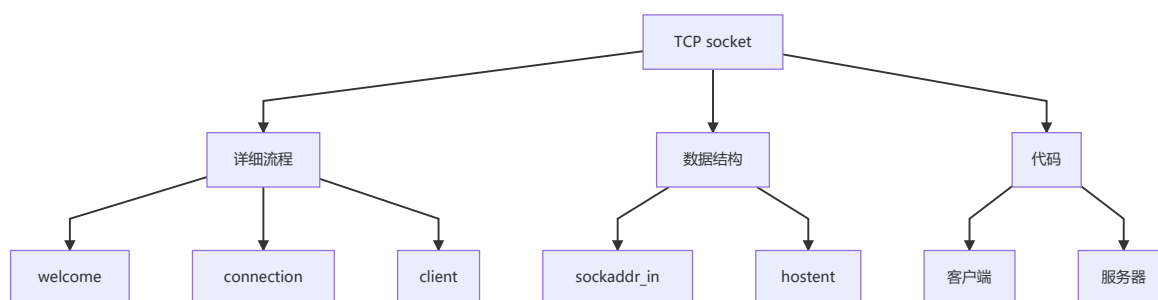
两个进程可以同时守候同一个端口。socket不一样代表不同的进程。

**数据结构：**两个结构体

- **sockaddr\_in**：端节点
  1. sin\_family：地址族（也可以用于IPX通讯，用于标识）
  2. sin\_port：端口号
  3. sin\_addr：IP地址
  4. sin\_zero[8]：对齐
- **hostent**：域名地址转换
  1. \*h\_name：主机域名（字符串指针）
  2. \*\*h\_aliases：主机一系列别名（二维）
  3. h\_length：地址长度（32位等）
  4. \*\*h\_addr\_list[i]：IP地址列表（域名解析函数调用）

**代码：**

- 客户端：**两个参数**（服务器域名argv[1]，服务器端口号argv[2]）argv[0]是应用程序本身的名字。创建结构体变量sad。端口号变成**短整型**（防止高低位出错）。**调用解析器**获取IP地址。**调用connect**函数建立连接（默认bind）。gets（输入）、write（发送）、read（接收）、printf（打印）、close（关闭）。
- 服务器：**一个参数**（当前应用进程守候的端口号）。两个结构体——sad本地端节点，cad客户端节点。端口变成整数。sad、cad赋值，转换成**短整型**。建立welcome socket。手动bind。死循环while(1)，read、write、close。**listen**（建立一个队列，来的连接请求进入排队，每次取一个进行服务）。也可以做成并发模式。

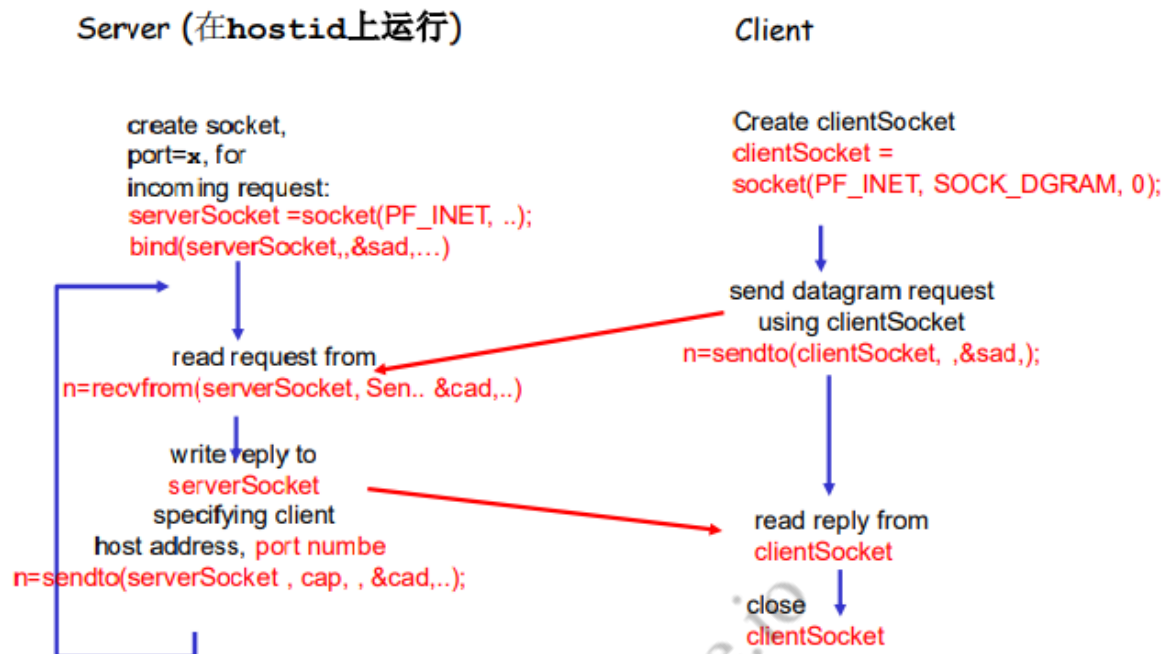


## 2.9 UDP 套接字编程

没有连接、没有握手，不和对方IP端口相捆绑。每次发送必须指明对方IP端口作为报文一部分。收的时候要知道内容和对方端节点IP端口。可能乱序，也可能丢失。——协议数据单元叫做数据报。

应用程序角度：UDP 为客户端和服务端提供**不可靠的字节组**的传送服务。

## 过程



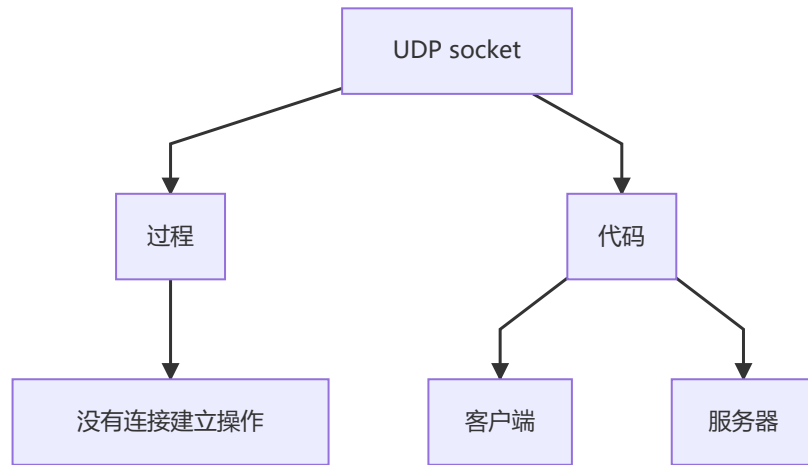
1. serverSocket **创建**起一个socket——服务器
2. bind 和本地IP+端口相**捆绑**。IP地址作为系统常量——服务器
3. recvFrom **等待**客户端请求，有值解除阻塞——服务器
4. clientSocket **创建**本地socket——客户端
5. 默认**捆绑**本地端口——客户端
6. sendto **发送**请求，指明socket值和内容——客户端
7. serverSocket sendTo (write) 解除阻塞，进行处理，**发送**内容——服务器
8. read **读取**内容进行显示——客户端
9. 服务器返回等待——没有连接关闭操作
10. close 关闭连接，删掉client socket——客户端

服务器端就一个socket，没有welcom和connection之分。

## 代码

- 客户端：两个参数-主机名+端口号。建立socket，sad变量清零赋对应的值。get获取输入、Send发送请求、read读取响应、关闭连接。
- 服务器：一个参数-守候端口号（转为整数）。sad赋值。和本地端口号相捆绑。死循环进行操作，接收请求、处理、进行发送。





## 2.10 小结

应用程序体系结构 1 客户-服务器 2 P2P 3 混合。

传输层服务质量：可靠性、带宽、延时、安全。

传输层服务模式：TCP、UDP。

各种流行的协议：☐ HTTP ☐ FTP ☐ SMTP, POP, IMAP ☐ DNS

Socket编程

**详细知识点：**

**应用层报文类型：**请求/响应报文。

**报文格式：**首部、数据。

**报文分类：**控制报文 vs. 数据报文、带内vs. 带外

**模式：**集中式 vs. 分散式、无状态 vs. 维护状态

**报文传输：**可靠的 vs. 不可靠的

总的一条：**在网络边缘处理复杂性业务。**