> **整理自公众号：Java专栏**
>
> 微信扫描二维码，后台回复【**导图**】，获取47张Java相关思维导图。
>
> 后台回复【**实战**】，获取5个最新微服务项目源码&&实战视屏教程（从基础环境到成功部署）

1.先看一天面试的经验：

第一场：

- 面试官：你说一下TCP的三次握手
- 我：第一次Client将SYN置1.....、第二次Server收.....、 第三次.....
- 面试官：很难背吧？
- 我：.....是啊，很难，要不我在和你说说四次挥手？
- 面试官：别了别了回去等通知吧.....
- 我："....."

第二场： `心里憋了一万个草泥马来到的第二家`

- ....
- 面试官：你说一下TCP的三次握手
- 我（`心里在想，还来？`）：没什么好说的，就是为了保持一次网络通信交互正常
- 面试官：你能说的清楚一点吗？
- 我：就等于是你在不认识我的情况下打我的电话让我来面试
- 面试官："懵了"，好像是这么回事
- 面试官：你说一下TCP的四次挥手
- 我：等于是我在上家公司辞职了
- 面试官："想了一下"，能不能说的清楚一点吗？
- 我：我找老板办理离职，老板说可以，老板接着给我办理离职，我才可以走
- 面试官：有道理！
- 面试官：你说一下TCP和UDP的区别吧
- 我：TCP等于和陌生人打电话处理事情，UDP等于发广播
- 面试官："..."有道理
- 面试官：你期望薪资多少
- 我：15K
- 面试官：下周一有时间入职？
- 我：....

接着进入正题

2.什么是网络编程

- 网络编程的本质是多台计算机之间的数据交换。数据传递本身没有多大的难度，不就是把一个设备中的数据发送给其他设备，然后接受另外一个设备反馈的数据。现在的网络编程基本上都是基于请求/响应方式的，也就是一个设备发送请求数据给另外一个，然后接收另一个设备的反馈。在网络编程中，发起连接程序，也就是发送第一次请求的程序，被称作客户端(Client)，等待其他程序连接的程序被称作服务器(Server)。客户端程序可以在需要的时候启动，而服务器为了能够时刻相应连接，则需要一直启动。
- 例如以打电话为例，首先拨号的人类似于客户端，接听电话的人必须保持电话畅通类似于服务器。连接一旦建立以后，就客户端和服务器端就可以进行数据传递了，而且两者的身份是等价的。在一些程序中，程序既有客户端功能也有服务器端功能，最常见的软件就是QQ、微信这类软件了。

3.网络编程中两个主要的问题

1. 一个是如何准确的定位网络上一台或多台主机，
2. 另一个就是找到主机后如何可靠高效的进行数据传输。

- 在TCP/IP协议中IP层主要负责网络主机的定位，数据传输的路由，由IP地址可以唯一地确定Internet上的一台主机。
- 而TCP层则提供面向应用的可靠（TCP）的或非可靠（UDP）的数据传输机制，这是网络编程的主要对象，一般不需要关心IP层是如何处理数据的。
- 目前较为流行的网络编程模型是客户机/服务器（C/S）结构。即通信双方一方作为服务器等待客户提出请求并予以响应。客户则在需要服务时向服务器提 出申请。服务器一般作为守护进程始终运行，监听网络端口，一旦有客户请求，就会启动一个服务进程来响应该客户，同时自己继续监听服务端口，使后来的客户也 能及时得到服务。

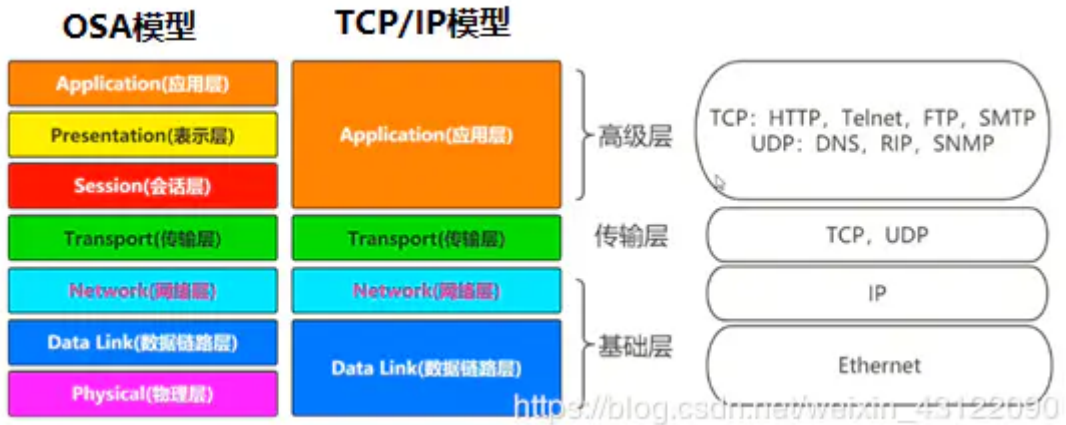
4.网络协议是什么

- 在计算机网络要做到井井有条的交换数据，就必须遵守一些事先约定好的规则，比如交换数据的格式、是否需要发送一个应答信息。这些规则被称为网络协议。

5.为什么要对网络协议分层

- 简化问题难度和复杂度。由于各层之间独立，我们可以分割大问题为小问题。
- 灵活性好。当其中一层的技术变化时，只要层间接口关系保持不变，其他层不受影响。
- 易于实现和维护。
- 促进标准化工作。分开后，每层功能可以相对简单地被描述

6.计算机网络体系结构



- **OSI参考模型**
  - OSI（Open System Interconnect），即开放式系统互联。一般都叫OSI参考模型，是ISO（国际标准化组织）组织在1985年研究的网络互连模型。ISO为了更好的使网络应用更为普及，推出了OSI参考模型，这样所有的公司都按照统一的标准来指定自己的网络，就可以互通互联了。
  - OSI定义了网络互连的七层框架（物理层、数据链路层、网络层、传输层、会话层、表示层、应用层）。

OSI七层模型	功能	对应的网络协议	TCP/IP四层概念模型
应用层	文件传输，文件管理，电子邮件的信息处理——apdu	HTTP、TFTP，FTP，NFS，WAIS、SMTP	应用层
表示层	确保一个系统的应用层发送的消息可以被另一个系统的应用层读取，编码转换，数据解析，管理数据的解密和加密，最小单位——ppdu	Telnet，Rlogin，SNMP，Gopher	
会话层	负责在网络中的两节点建立，维持和终止通信，在一层协议中，可以解决节点连接的协调和管理问题。包括通信连接的建立，保持会话过程通信连接的畅通，两节点之间的对话，决定通信是否被终端一斤通信终端是决定从何处重新发送，最小单位——spdu	SMTP，DNS	
传输层	定义一些传输数据的协议和端口。传输协议同时进行流量控制，或是根据接收方接收数据的快慢程度，规定适当的发送速率，解决传输效率及能力的问题——tpdu	TCP，UDP	传输层
网络层	控制子网的运行，如逻辑编址，分组传输，路由选择最小单位——分组（包）报文	IP，ICMP，ARP，RARP，AKP，UUCP	网络层
数据链路层	主要是对物理层传输的比特流包装，检测保证数据传输的可靠性，将物理层接收的数据进行MAC（媒体访问控制）地址的封装和解封装，也可以简单的理解为物理寻址。交换机就处在这一层，最小的传输单位——帧	FDDI，Ethernet，Arpanet，PDN，SLIP，PPP，STP。HDLC，SDLC，帧中继	数据链路层
物理层	定义物理设备的标准，主要对物理连接方式，电气特性，机械特性等制定统一标准，传输比特流，因此最小的传输单位——位（比特流）	IEEE 802.1A，IEEE 802.2到IEEE 802.	

- **TCP/IP参考模型**
- TCP/IP四层协议（数据链路层、网络层、传输层、应用层）

1. 应用层 应用层最靠近用户的一层，是为计算机用户提供应用接口，也为用户直接提供各种网络服务。我们常见应用层的网络服务协议有：HTTP，HTTPS，FTP，TELNET等。
2. 传输层 建立了主机端到端的链接，传输层的作用是为上层协议提供端到端的可靠和透明的数据传输服务，包括处理差错控制和流量控制等问题。该层向高层屏蔽了下层数据通信的细节，使高层用户看到的只是在两个传输实体间的一条主机到主机的、可由用户控制和设定的、可靠的数据通路。我们通常说的，TCP UDP就是在这一层。端口号既是这里的"端"。
3. 网络层 本层通过IP寻址来建立两个节点之间的连接，为源端的运输层送来的分组，选择合适的路由和交换节点，正确无误地按照地址传送给目的端的运输层。就是通常说的IP层。这一层就是我们经常说的IP协议层。IP协议是Internet的基础。
4. 数据链路层 通过一些规程或协议来控制这些数据的传输，以保证被传输数据的正确性。实现这些规程或协议的`硬件`和软件加到物理线路，这样就构成了数据链路，

1 TCP / UDP

1.1 什么是TCP/IP和UDP

- TCP/IP即传输控制/网络协议，是面向连接的协议，发送数据前要先建立连接(发送方和接收方的成对的两个之间必须建 立连接)，TCP提供可靠的服务，也就是说，通过TCP连接传输的数据不会丢失，没有重复，并且按顺序到达
- UDP它是属于TCP/IP协议族中的一种。是无连接的协议，发送数据前不需要建立连接，是没有可靠性的协议。因为不需要建立连接所以可以在在网络上以任何可能的路径传输，因此能否到达目的地，到达目的地的时间以及内容的正确性都是不能被保证的。

1.2 TCP与UDP区别：

- TCP是面向连接的协议，发送数据前要先建立连接，TCP提供可靠的服务，也就是说，通过TCP连接传输的数据不会丢失，没有重复，并且按顺序到达；
- UDP是无连接的协议，发送数据前不需要建立连接，是没有可靠性；
- TCP通信类似于于要打个电话，接通了，确认身份后，才开始进行通行；
- UDP通信类似于学校广播，靠着广播播报直接进行通信。
- TCP只支持点对点通信，UDP支持一对一、一对多、多对一、多对多；
- TCP是面向字节流的，UDP是面向报文的； 面向字节流是指发送数据时以字节为单位，一个数据包可以拆分成若干组进行发送，而UDP一个报文只能一次发完。
- TCP首部开销（20字节）比UDP首部开销（8字节）要大
- UDP 的主机不需要维持复杂的连接状态表

1.3 TCP和UDP的应用场景：

- 对某些实时性要求比较高的情况使用UDP，比如游戏，媒体通信，实时直播，即使出现传输错误也可以容忍；其它大部分情况下，HTTP都是用TCP，因为要求传输的内容可靠，不出现丢失的情况

1.4 形容一下TCP和UDP

- **TCP通信可看作打电话：**

  李三(拨了个号码)：喂，是王五吗？ 王五：哎，您谁啊？ 李三：我是李三，我想给你说点事儿，你现在方便吗？ 王五：哦，我现在方便，你说吧。 甲：那我说了啊？ 乙：你说吧。 (连接建立了，接下来就是说正事了...)

- **UDP通信可看为学校里的广播：**

  播音室：喂喂喂！全体操场集合

1.5 运行在TCP 或UDP的应用层协议分析。

- 运行在TCP协议上的协议：
  - HTTP（Hypertext Transfer Protocol，超文本传输协议），主要用于普通浏览。
  - HTTPS（HTTP over SSL，安全超文本传输协议）,HTTP协议的安全版本。
  - FTP（File Transfer Protocol，文件传输协议），用于文件传输。
  - POP3（Post Office Protocol, version 3，邮局协议），收邮件用。
  - SMTP（Simple Mail Transfer Protocol，简单邮件传输协议），用来发送电子邮件。
  - TELNET（Teletype over the Network，网络电传），通过一个终端（terminal）登陆到网络。
  - SSH（Secure Shell，用于替代安全性差的TELNET），用于加密安全登陆用。
- 运行在UDP协议上的协议：
  - BOOTP（Boot Protocol，启动协议），应用于无盘设备。
  - NTP（Network Time Protocol，网络时间协议），用于网络同步。
  - DHCP（Dynamic Host Configuration Protocol，动态主机配置协议），动态配置IP地址。
- 运行在TCP和UDP协议上：

- DNS（Domain Name Service，域名服务），用于完成地址查找，邮件转发等工作。
- ECHO（Echo Protocol，回绕协议），用于查错及测量应答时间（运行在TCP和UDP协议上）。
- SNMP（Simple Network Management Protocol，简单网络管理协议），用于网络信息的收集和网络管理。
- DHCP（Dynamic Host Configuration Protocol，动态主机配置协议），动态配置IP地址。
- ARP（Address Resolution Protocol，地址解析协议），用于动态解析以太网硬件的地址。

什么是ARP协议 (Address Resolution Protocol)?

- **ARP协议完成了IP地址与物理地址的映射。**每一个主机都设有一个 ARP 高速缓存，里面有**所在的局域网**上的各主机和路由器的 IP 地址到硬件地址的映射表。当源主机要发送数据包到目的主机时，会先检查自己的ARP高速缓存中有没有目的主机的MAC地址，如果有，就直接将数据包发到这个MAC地址，如果没有，就向**所在的局域网**发起一个ARP请求的广播包（在发送自己的 ARP 请求时，同时会带上自己的 IP 地址到硬件地址的映射），收到请求的主机检查自己的IP地址和目的主机的IP地址是否一致，如果一致，则先保存源主机的映射到自己的ARP缓存，然后给源主机发送一个ARP响应数据包。源主机收到响应数据包之后，先添加目的主机的IP地址与MAC地址的映射，再进行数据传送。如果源主机一直没有收到响应，表示ARP查询失败。
- 如果所要找的主机和源主机不在同一个局域网上，那么就要通过 ARP 找到一个位于本局域网上的某个路由器的硬件地址，然后把分组发送给这个路由器，让这个路由器把分组转发给下一个网络。剩下的工作就由下一个网络来做。

什么是NAT (Network Address Translation, 网络地址转换)?

- 用于解决内网中的主机要和因特网上的主机通信。由NAT路由器将主机的本地IP地址转换为全球IP地址，分为静态转换（转换得到的全球IP地址固定不变）和动态NAT转换。

从输入址到获得页面的过程?

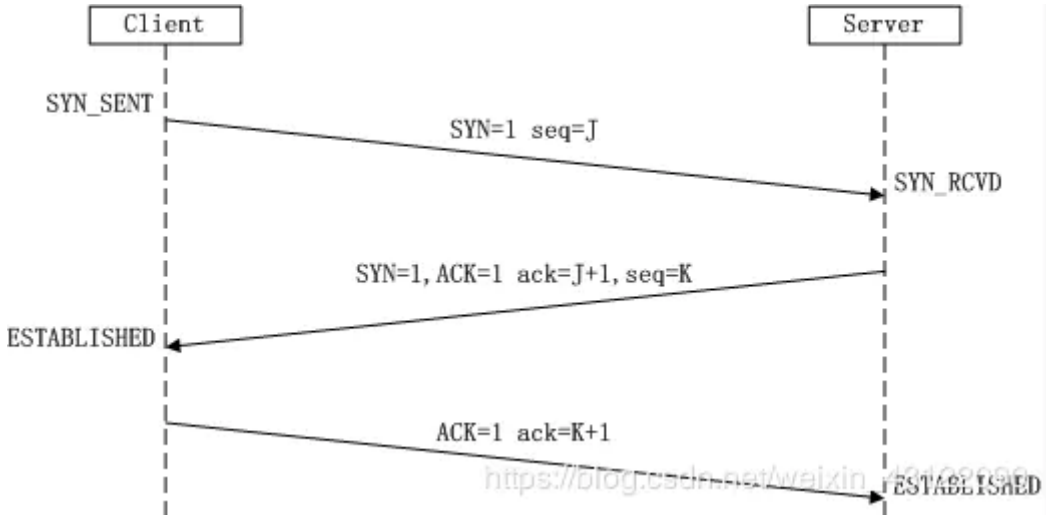
1. 浏览器查询 DNS，获取域名对应的IP地址:具体过程包括浏览器搜索自身的DNS缓存、搜索操作系统的DNS缓存、读取本地的Host文件和向本地DNS服务器进行查询等。对于向本地DNS服务器进行查询，如果要查询的域名包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析(此解析具有权威性)；如果要查询的域名不由本地DNS服务器区域解析，但该服务器已缓存了此网址映射关系，则调用这个IP地址映射，完成域名解析（此解析不具有权威性）。如果本地域名服务器并未缓存该网址映射关系，那么将根据其设置发起递归查询或者迭代查询；
2. 浏览器获得域名对应的IP地址以后，浏览器向服务器请求建立链接，发起三次握手；
3. TCP/IP链接建立起来后，浏览器向服务器发送HTTP请求；
4. 服务器接收到这个请求，并根据路径参数映射到特定的请求处理器进行处理，并将处理结果及相应的视图返回给浏览器；
5. 浏览器解析并渲染视图，若遇到对js文件、css文件及图片等静态资源的引用，则重复上述步骤并向服务器请求这些资源；
6. 浏览器根据其请求到的资源、数据渲染页面，最终向用户呈现一个完整的页面。

1.6 TCP的三次握手

1.6.1 什么是TCP的三次握手

- 在网络数据传输中，传输层协议TCP是要建立连接的可靠传输，TCP建立连接的过程，我们称为三次握手。

1.6.2 三次握手的具体细节



1. 第一次握手：Client将SYN置1，随机产生一个初始序列号seq发送给Server，进入SYN_SENT状态；
2. 第二次握手：Server收到Client的SYN=1之后，知道客户端请求建立连接，将自己的SYN置1，ACK置1，产生一个acknowledge number=sequence number+1，并随机产生一个自己的初始序列号，发送给客户端；进入SYN_RCVD状态；
3. 第三次握手：客户端检查acknowledge number是否为序列号+1，ACK是否为1，检查正确之后将自己的ACK置为1，产生一个acknowledge number=服务器发的序列号+1，发送给服务器；进入ESTABLISHED状态；服务器检查ACK为1和acknowledge number为序列号+1之后，也进入ESTABLISHED状态；完成三次握手，连接建立。

- 简单来说就是：
    1. 客户端向服务端发送SYN
    2. 服务端返回SYN,ACK
    3. 客户端发送ACK

1.6.3 用现实理解三次握手的具体细节

- 三次握手的目的是建立可靠的通信信道，主要的目的就是双方确认自己与对方的发送与接收机能正常。

1. 第一次握手：客户什么都不能确认；服务器确认了对方发送正常
2. 第二次握手：客户确认了：自己发送、接收正常，对方发送、接收正常；服务器确认 了：自己接收正常，对方发送正常
3. 第三次握手：客户确认了：自己发送、接收正常，对方发送、接收正常；服务器确认 了：自己发送、接收正常，对方发送接收正常 所以三次握手就能确认双发收发功能都正常，缺一不可。

1.6.4 建立连接可以两次握手吗？为什么?

- 不可以。
- 因为可能会出现已失效的连接请求报文段又传到了服务器端。 > client 发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达 server。本来这是一个早已失效的报文段。但 server 收到此失效的连接请求报文段后，就误认为是 client 再次发出的一个新的连接请求。于是就向 client 发出确认报文段，同意建立连接。假设不采用 “三次握手”，那么只要 server 发出确认，新的连接就建立了。由于现在 client 并没有发出建立连接的请求，因此不会理睬 server 的确认，也不会向 server 发送数据。但 server 却以为新的运输连接已经建立，并一直等待 client 发来数据。这样，server 的很多资源就白白浪费掉了。采用 “三次握手” 的办法可以防止上述现象发生。例如刚才那种情况，client 不会向 server 的确认发出确认。server 由于收不到确认，就知道 client 并没有要求建立连接。
- 而且，两次握手无法保证Client正确接收第二次握手的报文（Server无法确认Client是否收到），也无法保证Client和Server之间成功互换初始序列号。

1.6.5 可以采用四次握手吗？为什么？

- 这个肯定可以。三次握手都可以保证连接成功了，何况是四次，但是会降低传输的效率。

1.6.6 第三次握手中，如果客户端的ACK未送达服务器，会怎样？

- Server端：由于Server没有收到ACK确认，因此会每隔 3秒 重发之前的SYN+ACK（默认重发五次，之后自动关闭连接进入CLOSED状态），Client收到后会重新传ACK给Server。
- Client端，会出现两种情况：
    1. 在Server进行超时重发的过程中，如果Client向服务器发送数据，数据头部的ACK是为1的，所以服务器收到数据之后会读取 ACK number，进入 establish 状态
    2. 在Server进入CLOSED状态之后，如果Client向服务器发送数据，服务器会以RST包应答。

1.6.7 如果已经建立了连接，但客户端出现了故障怎么办？

- 服务器每收到一次客户端的请求后都会重新复位一个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒钟发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

1.6.8 初始序列号是什么？

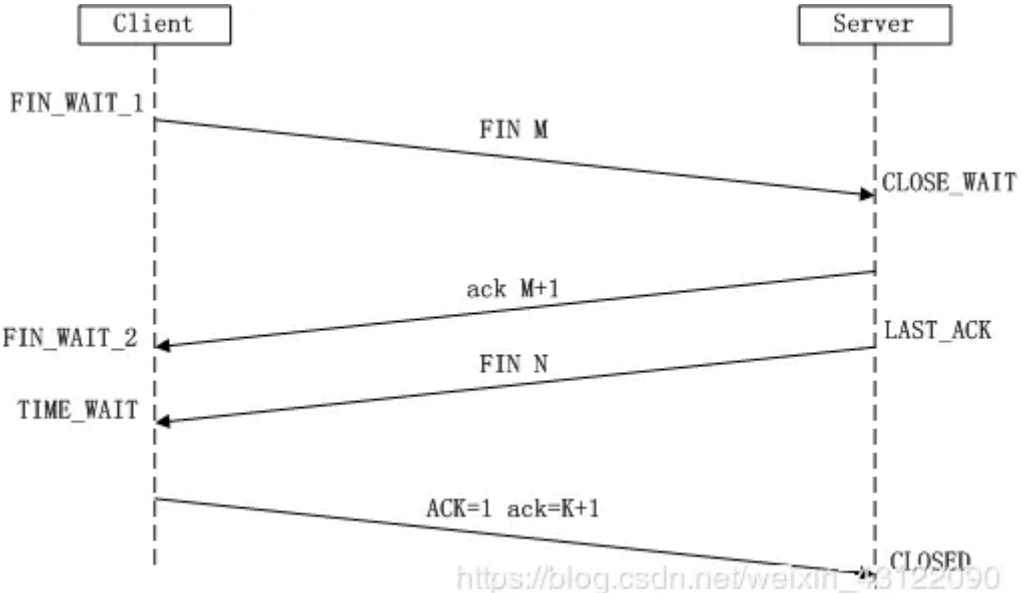
- TCP连接的一方A，随机选择一个32位的序列号（Sequence Number）作为发送数据的初始序列号（Initial Sequence Number，ISN），比如为1000，以该序列号为原点，对要传送的数据进行编号：1001、1002...三次握手时，把这个初始序列号传送给另一方B，以便在传输数据时，B可以确认什么样的数据编号是合法的；同时在进行数据传输时，A还可以确认B收到的每一个字节，如果A收到了B的确认编号（acknowledge number）是2001，就说明编号为1001-2000的数据已经被B成功接受。

1.7 TCP的四次挥手

1.7.1 什么是TCP的四次挥手

- 在网络数据传输中，传输层协议断开连接的过程我们称为四次挥手

1.7.2 四次挥手的具体细节



1. 第一次挥手：Client将FIN置为1，发送一个序列号seq给Server；进入FIN_WAIT_1状态；
2. 第二次挥手：Server收到FIN之后，发送一个ACK=1，acknowledge number=收到的序列号+1；进入CLOSE_WAIT状态。此时客户端已经没有要发送的数据了，但仍可以接受服务器发来的数据。
3. 第三次挥手：Server将FIN置1，发送一个序列号给Client；进入LAST_ACK状态；

4. 第四次挥手：Client收到服务器的FIN后，进入TIME_WAIT状态；接着将ACK置1，发送一个acknowledge number=序列号+1给服务器；服务器收到后，确认acknowledge number后，变为CLOSED状态，不再向客户端发送数据。客户端等待2*MSL（报文段最长寿命）时间后，也进入CLOSED状态。完成四次挥手。

1.7.3 用现实理解三次握手的具体细节TCP的四次挥手

- 四次挥手断开连接是因为要确定数据全部传书完了

1. 客户与服务器交谈结束之后，客户要结束此次会话，就会对服务器说：我要关闭连接了（第一 次挥手）
2. 服务器收到客户的消息后说：好的，你要关闭连接了。（第二次挥手）
3. 然后服务器确定了没有话要和客户说了，服务器就会对客户说，我要关闭连接了。(第三次挥 手)
4. 客户收到服务器要结束连接的消息后说：已收到你要关闭连接的消息。(第四次挥手)，才关闭

1.7.4 为什么不能把服务器发送的ACK和FIN合并起来，变成三次挥手（CLOSE_WAIT状态意义是什么）？

- 因为服务器收到客户端断开连接的请求时，可能还有一些数据没有发完，这时先回复ACK，表示接收到了断开连接的请求。等到数据发完之后再发FIN，断开服务器到客户端的数据传送。

1.7.5 如果第二次挥手时服务器的ACK没有送达客户端，会怎样?

- 客户端没有收到ACK确认，会重新发送FIN请求。

1.7.6 客户端TIME_WAIT状态的意义是什么?

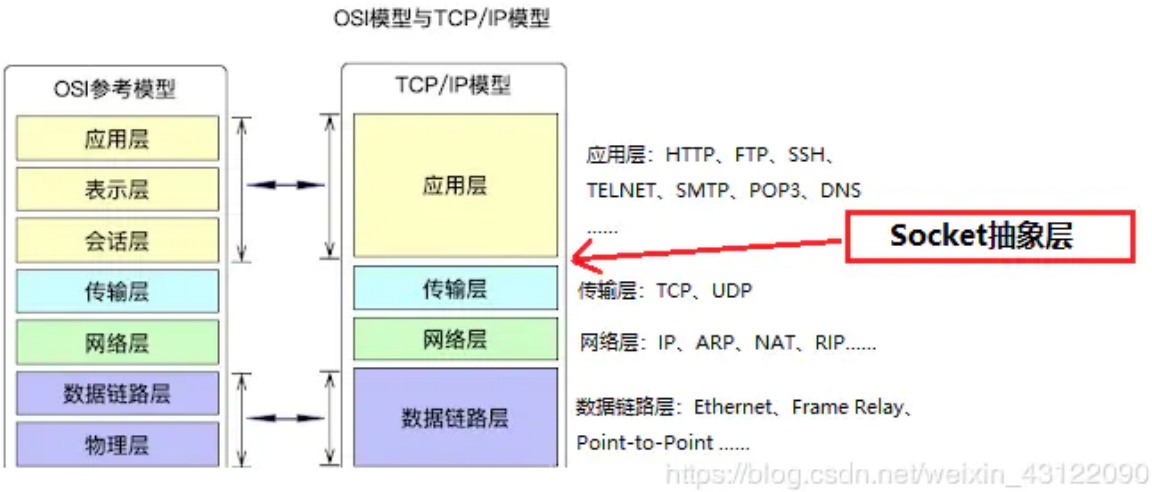
- 第四次挥手时，客户端发送给服务器的ACK有可能丢失，TIME_WAIT状态就是用来重发可能丢失的ACK报文。如果Server没有收到ACK，就会重发FIN，如果Client在2*MSL的时间内收到了FIN，就会重新发送ACK并再次等待2MSL，防止Server没有收到ACK而不断重发FIN。 MSL(Maximum Segment Lifetime)，指一个片段在网络中最大的存活时间，2MSL就是一个发送和一个回复所需的最大时间。如果直到2MSL，Client都没有再次收到FIN，那么Client推断ACK已经被成功接收，则结束TCP连接。

2 Socket

1 什么是Socket

- 网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个Socket。Socket通常用来实现客户方和服务方的连接。Socket是TCP/IP协议的一个十分流行的编程界面，一个Socket由一个IP地址和一个端口号唯一确定。
- 但是，Socket所支持的协议种类也不光TCP/IP、UDP，因此两者之间是没有必然联系的。在Java环境下，Socket编程主要是指基于TCP/IP协议的网络编程。
- socket连接就是所谓的长连接，客户端和服务器需要互相连接，理论上客户端和服务器端一旦建立起连接将不会主动断掉的，但是有时候网络波动还是有可能的
- Socket偏向于底层。一般很少直接使用Socket来编程，框架底层使用Socket比较多，

2 socket属于网络的那个层面



- Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个外观模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议。

3 Socket通讯的过程

- 基于TCP：服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。
- 基于UDP：UDP 协议是用户数据报协议的简称，也用于网络数据的传输。虽然 UDP 协议是一种不太可靠的协议，但有时在需要较快地接收数据并且可以忍受较小错误的情况下，UDP 就会表现出更大的优势。我客户端只需要发送，服务端能不能接收的到我不管

4 TCP协议Socket代码示例：

`先运行服务端，在运行客户端`，

1. 服务端：

```
package com.test.io;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
```

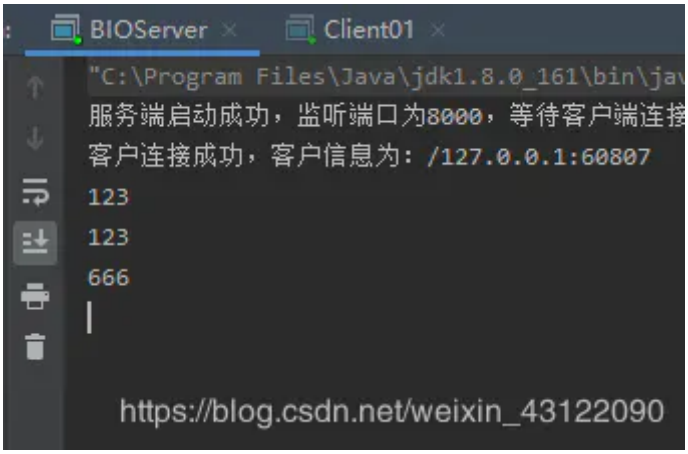
```
import java.net.ServerSocket;
import java.net.Socket;

//TCP协议Socket使用BIO进行通行：服务端
public class BIOServer {
    // 在main线程中执行下面这些代码
  public static void main(String[] args) {
        //1单线程服务
  ServerSocket server = null;
        Socket socket = null;
        InputStream in = null;
        OutputStream out = null;
        try {
            server = new ServerSocket(8000);
            System.out.println("服务端启动成功，监听端口为8000，等待客户端连接...");
            while (true){
                socket = server.accept(); //等待客户端连接
  System.out.println("客户连接成功，客户信息为：" +
socket.getRemoteSocketAddress());
                in = socket.getInputStream();
                byte[] buffer = new byte[1024];
                int len = 0;
                //读取客户端的数据
  while ((len = in.read(buffer)) > 0) {
                    System.out.println(new String(buffer, 0, len));
                }
                //向客户端写数据
  out = socket.getOutputStream();
                out.write("hello!".getBytes());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

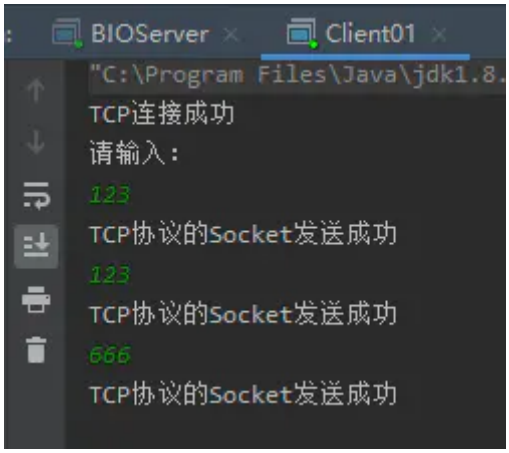
1. 客户端：

```
package com.test.io;

import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import java.util.Scanner;

//TCP协议Socket：客户端
public class Client01 {
    public static void main(String[] args) throws IOException {
        //创建套接字对象socket并封装ip与port
        Socket socket = new Socket("127.0.0.1", 8000);
        //根据创建的socket对象获得一个输出流
  OutputStream outputStream = socket.getOutputStream();
        //控制台输入以IO的形式发送到服务器
  System.out.println("TCP连接成功 \n请输入：");
        while(true){
            byte[] car = new Scanner(System.in).nextLine().getBytes();
```

```
            outputStream.write(car);
            System.out.println("TCP协议的Socket发送成功");
            //刷新缓冲区
  outputStream.flush();
        }
    }
}
```

`先运行服务端，在运行客户端`。测试结果发送成功：





5 UDP协议Socket代码示例：

`先运行服务端，在运行客户端`

1. 服务端：

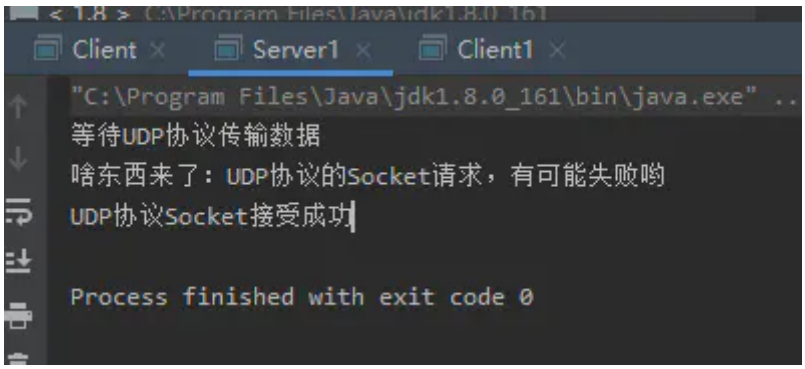
```
//UDP协议Socket：服务端
public class Server1 {
    public static void main(String[] args) {
        try {
            //DatagramSocket代表声明一个UDP协议的Socket
            DatagramSocket socket = new DatagramSocket(8888);
            //byte数组用于数据存储。
  byte[] car = new byte[1024];
            //DatagramPacket 类用来表示数据报包DatagramPacket
            DatagramPacket packet = new DatagramPacket(car, car.length);
            // //创建DatagramPacket的receive()方法来进行数据的接收,等待接收一个socket请
求后才执行后续操作;
  System.out.println("等待UDP协议传输数据");
            socket.receive(packet);
            //packet.getLength返回将要发送或者接收的数据的长度。
```

```
  int length = packet.getLength();
            System.out.println("啥东西来了：" + new String(car, 0, length));
            socket.close();
            System.out.println("UDP协议Socket接受成功");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

1. 客户端：

```
//UDP协议Socket：客户端
public class Client1 {
    public static void main(String[] args) {
        try {
            //DatagramSocket代表声明一个UDP协议的Socket
            DatagramSocket socket = new DatagramSocket(2468);
            //字符串存储人Byte数组
  byte[] car = "UDP协议的Socket请求，有可能失败哟".getBytes();
            //InetSocketAddress类主要作用是封装端口
  InetSocketAddress address = new InetSocketAddress("127.0.0.1", 8888);
            //DatagramPacket 类用来表示数据报包DatagramPacket
            DatagramPacket packet = new DatagramPacket(car, car.length,
address);
            //send() 方法发送数据包。
  socket.send(packet);
            System.out.println("UDP协议的Socket发送成功");
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

`先运行服务端，在运行客户端`。测试结果成功发送成功：



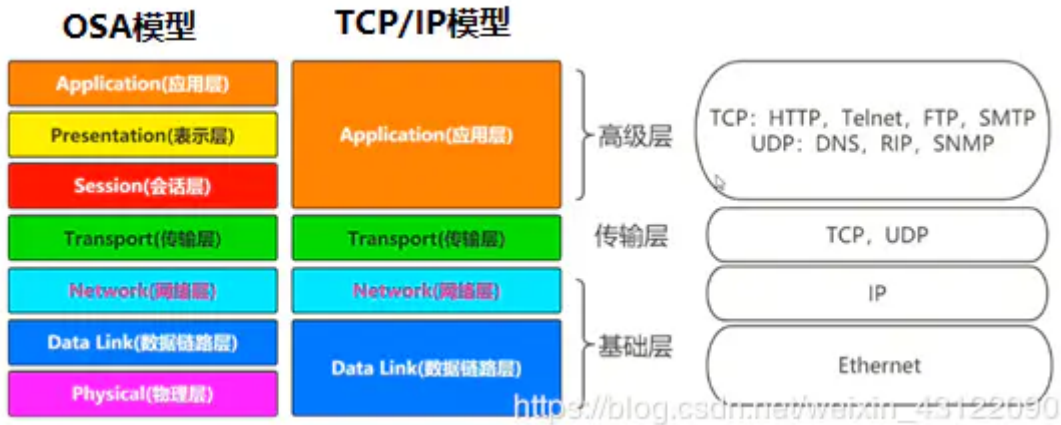
6 Socket的常用类

类名	用于	作用
Socket	TCP协议	Socket类同时工作于客户端和服务端，所有方法都是通用的，这个类三个主要作用，校验包信息，发起连接（Client），操作流数据（Client/Server）
ServerSocket	TCP协议	ServerSocket表示为服务端，主要作用就是绑定并监听一个服务器端口，为每个建立连接的客户端“克隆/映射”一个Socket对象，具体数据操作都是通过这个Socket对象完成的，ServerSocket只关注如何和客户端建立连接
DatagramSocket	ODP协议	DatagramSocket 类用于表示发送和接收数据报包的套接字。
DatagramPacket	ODP协议	DatagramPacket 类用来表示数据报包，数据报包用来实现无连接包投递服务。
InetAddress	IP+端口号	Java提供了InetAddress类来代表互联网协议（IP）地址，InetAddress类没有提供构造器，而是提供了如下两个静态方法来获取InetAddress实例：
InetSocketAddress	IP+端口号	在使用Socket来连接服务器时最简单的方式就是直接使用IP和端口，但Socket类中并未提供这种方式，而是靠SocketAddress的子类InetSocketAddress来实现 IP 地址 + 端口号的创建，不依赖任何协议。

3. HTTP

什么是Http协议?

- Http协议是对客户端和服务器端之间数据之间实现可靠性的传输文字、图片、音频、视频等超文本数据的规范，格式简称为“超文本传输协议”
- Http协议属于应用层，及用户访问的第一层就是http



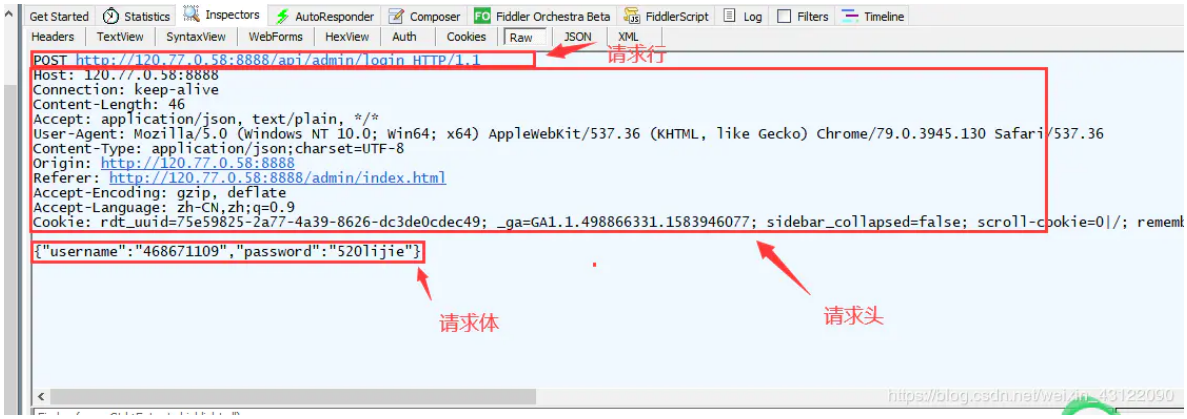
Socket和http的区别和应用场景

- Socket连接就是所谓的长连接，理论上客户端和服务器端一旦建立起连接将不会主动断掉；
- Socket适用场景：网络游戏，银行持续交互，直播，在线视屏等。
- http连接就是所谓的短连接，即客户端向服务器端发送一次请求，服务器端响应后连接即会断开等待下次连接
- http适用场景：公司OA服务，互联网服务，电商，办公，网站等等等等

什么是http的请求体?

- HTTP请求体是我们请求数据时先发送给服务器的数据，毕竟我向服务器那数据，先要表明我要什么吧
- HTTP请求体由：请求行 、请求头、请求数据组成的，
- 注意：GIT请求是没有请求体的

1. POST请求



2. GIT请求是没有请求体的

http的响应报文有哪些?

- http的响应报是服务器返回给我们的数据，必须先有请求体再有响应报文
- 响应报文包含三部分 状态行、响应首部字段、响应内容实体实现

http和https的区别?

- 其实HTTPS就是从HTTP加上加密处理（一般是SSL安全通信线路）+认证+完整性保护
- 区别：
    1. http需要拿到ca证书，需要钱的
    2. 端口不一样，http是80，https443
    3. http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。
    4. http和https使用的是完全不同的连接方式（http的连接很简单，是无状态的；HTTPS 协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。）

HTTPS工作原理

- 一、首先HTTP请求服务端生成证书，客户端对证书的有效期、合法性、域名是否与请求的域名一致、证书的公钥（RSA加密）等进行校验；
- 二、客户端如果校验通过后，就根据证书的公钥的有效， 生成随机数，随机数使用公钥进行加密（RSA加密）；
- 三、消息体产生的后，对它的摘要进行MD5（或者SHA1）算法加密，此时就得到了RSA签名；
- 四、发送给服务端，此时只有服务端（RSA私钥）能解密。
- 五、解密得到的随机数，再用AES加密，作为密钥（此时的密钥只有客户端和服务端知道）。

一次完整的HTTP请求所经历几个步骤?

`HTTP通信机制是在一次完整的HTTP通信过程中，Web浏览器与Web服务器之间将完成下列7个步骤：`

1. 建立TCP连接

怎么建立连接的，看上面的三次捂手

2. Web浏览器向Web服务器发送请求行

一旦建立了TCP连接，**Web浏览器就会向Web服务器发送请求命令。**例如：GET /sample/hello.jsp HTTP/1.1。

3. Web浏览器发送请求头

浏览器发送其请求命令之后，还要以头信息的形式向Web服务器发送一些别的信息，**之后浏览器发送了一空白行来通知服务器**，它已经结束了该头信息的发送。

4. Web服务器应答

客户机向服务器发出请求后，服务器会客户机回送应答， **HTTP/1.1 200 OK ，应答的第一部分是协议的版本号和应答状态码。**

5. Web服务器发送应答头

正如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档。

6. Web服务器向浏览器发送数据

Web服务器向浏览器发送头信息后，它会发送一个空白行来表示头信息的发送到此为结束，接着，**它就以Content-Type应答头信息所描述的格式发送用户所请求的实际数据。**

7. Web服务器关闭TCP连接

常用HTTP状态码是怎么分类的，有哪些常见的状态码？

- HTTP状态码表示客户端HTTP请求的返回结果、标识服务器处理是否正常、表明请求出现的错误等。
- 状态码的类别：

类别	描述
1xx：	指示信息–表示请求已接收，正在处理
2xx：	成功–表示请求已被成功接收、理解、接受
3xx：	重定向–要完成请求必须进行更进一步的操作
4xx：	客户端错误–请求有语法错误或请求无法实现
5xx：	服务器端错误–服务器未能实现合法的请求

- 常见的状态码：

状态码	描述
200：	请求被正常处理
204：	请求被受理但没有资源可以返回
206：	客户端只是请求资源的一部分，服务器只对请求的部分资源执行GET方法，相应报文中通过Content-Range指定范围的资源。
301：	永久性重定向
302：	临时重定向
303：	与302状态码有相似功能，只是它希望客户端在请求一个URI的时候，能通过GET方法重定向到另一个URI上
304：	发送附带条件的请求时，条件不满足时返回，与重定向无关
307：	临时重定向，与302类似，只是强制要求使用POST方法
400：	请求报文语法有误，服务器无法识别
401：	请求需要认证
403：	请求的对应资源禁止被访问
404：	服务器无法找到对应资源
500：	服务器内部错误
503：	服务器正忙

Http协议中有那些请求方式

请求方式	描述
GET：	用于请求访问已经被URI（统一资源标识符）识别的资源，可以通过URL传参给服务器
POST：	用于传输信息给服务器，主要功能与GET方法类似，但一般推荐使用POST方式。
PUT：	传输文件，报文主体中包含文件内容，保存到对应URI位置。
HEAD：	获得报文首部，与GET方法类似，只是不返回报文主体，一般用于验证URI是否有 > 效。
PATCH：	客户端向服务器传送的数据取代指定的文档的内容(部分取代)
TRACE：	回显客户端请求服务器的原始请求报文，用于"回环"诊断
DELETE：	删除文件，与PUT方法相反，删除对应URI位置的文件。
OPTIONS：	查询相应URI支持的HTTP方法。

GET方法与POST方法的区别

- **区别一**： get重点在从服务器上获取资源，post重点在向服务器发送数据；

- **区别二**： Get传输的数据量小，因为受URL长度限制，但效率较高； Post可以传输大量数据，所以上传文件时只能用Post方式；
- **区别三**： get是不安全的，因为get请求发送数据是在URL上，是可见的，可能会泄露私密信息，如密码等； post是放在请求头部的，是安全的

http版本的对比

- HTTP1.0版本的特性：
  - 早先1.0的HTTP版本，是一种无状态、无连接的应用层协议。
  - HTTP1.0规定浏览器和服务器保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器处理完成后立即断开TCP连接（无连接），服务器不跟踪每个客户端也不记录过去的请求（无状态）。
- HTTP1.1版本新特性
  - 默认持久连接节省通信量，只要客户端服务端任意一端没有明确提出断开TCP连接，就一直保持连接，可以发送多次HTTP请求
  - 管线化，客户端可以同时发出多个HTTP请求，而不用一个个等待响应
  - 断点续传原理
- HTTP2.0版本的特性
  - 二进制分帧（采用二进制格式的编码将其封装）
  - 首部压缩（设置了专门的首部压缩设计的HPACK算法。）
  - 流量控制（设置了接收某个数据流的多少字节一些流量控制）
  - 多路复用（可以在共享TCP链接的基础上同时发送请求和响应）
  - 请求优先级（可以通过优化这些帧的交错和传输顺序进一步优化性能）
  - 服务器推送（就是服务器可以对一个客户端请求发送多个响应。服务器向客户端推送资 源无需客户端明确的请求。（重大更新））

什么是对称加密与非对称加密

- 对称密钥加密是指加密和解密使用同一个密钥的方式，这种方式存在的最大问题就是密钥发送问题，即如何安全地将密钥发给对方；
- 而非对称加密是指使用一对非对称密钥，即公钥和私钥，公钥可以随意发布，但私钥只有自己知道。发送密文的一方使用对方的公钥进行加密处理，对方接收到加密信息后，使用自己的私钥进行解密。 由于非对称加密的方式不需要发送用来解密的私钥，所以可以保证安全性；但是和对称加密比起来，非常的慢

cookie和session对于HTTP有什么用?

- HTTP协议本身是无法判断用户身份。所以需要cookie或者session

什么是cookie

- cookie是由Web服务器保存在用户浏览器上的文件（key-value格式），可以包含用户相关的信息。客户端向服务器发起请求，就提取浏览器中的用户信息由http发送给服务器

什么是session

- session 是浏览器和服务器会话过程中，服务器会分配的一块储存空间给session。
- 服务器默认为客户浏览器的cookie中设置 sessionid，这个sessionid就和cookie对应，浏览器在向服务器请求过程中传输的cookie 包含 sessionid ，服务器根据传输cookie 中的 sessionid 获取出会话中存储的信息，然后确定会话的身份信息。

cookie与session区别

1. cookie数据存放在客户端上，安全性较差，session数据放在服务器上，安全性相对更高

2. 单个cookie保存的数据不能超过4K，session无此限制 信息后，使用自己的私钥进行解密。 由于非对称加密的方式不需要发送用来解密的私钥，所以可以保证安全性；但是和对称加密比起来，非常的慢

cookie和session对于HTTP有什么用?

- HTTP协议本身是无法判断用户身份。所以需要cookie或者session

什么是cookie

- cookie是由Web服务器保存在用户浏览器上的文件（key-value格式），可以包含用户相关的信息。客户端向服务器发起请求，就提取浏览器中的用户信息由http发送给服务器

什么是session

- session 是浏览器和服务器会话过程中，服务器会分配的一块储存空间给session。
- 服务器默认为客户浏览器的cookie中设置 sessionid，这个sessionid就和cookie对应，浏览器在向服务器请求过程中传输的cookie 包含 sessionid ，服务器根据传输cookie 中的 sessionid 获取出会话中存储的信息，然后确定会话的身份信息。

cookie与session区别

1. cookie数据存放在客户端上，安全性较差，session数据放在服务器上，安全性相对更高
2. 单个cookie保存的数据不能超过4K，session无此限制
3. session一定时间内保存在服务器上，当访问增多，占用服务器性能，考虑到服务器性能方面，应当使用cookie。