# 三种IO介绍

主要有三种IO模型：BIO、NIO、AIO

**BIO、NIO、AIO代码demo：（网上找）**

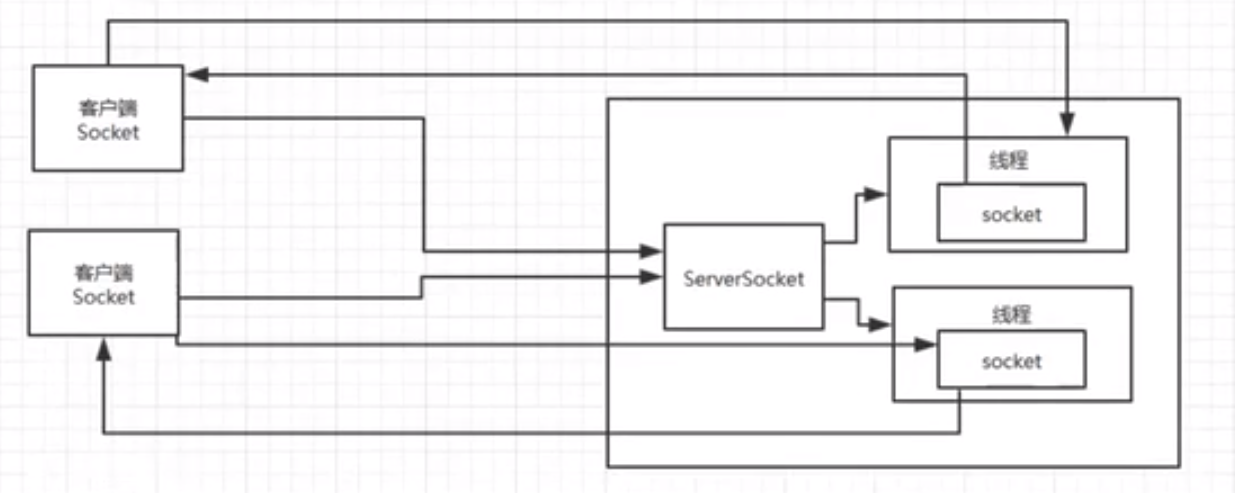
## BIO

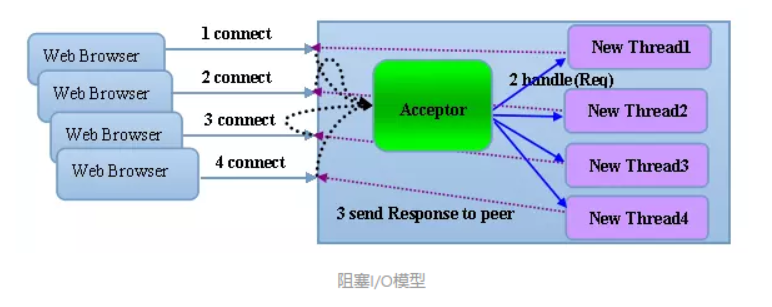
BIO是最传统的网络通信协议，同步阻塞式IO。就是服务端创建一个ServerSocket，然后客户端用一个Socket去连接该ServerSocket，然后ServerSocket接收到一个Socket的连接请求就创建一个Socket和一个线程去跟客户端Socket进行通信。

客户端和服务端的socket进行同步阻塞式的通信，客户端socket发送一个请求，服务端socket进行处理后返回响应，响应必须是等处理完后才会返回，在此之前啥事也干不了，这就是同步。

这个坑在于，每次一个客户端接入，都是要在服务端创建一个线程来服务这个客户端的，这会导致服务端的线程数量过大，而导致服务端程序的负载过高，最后宕机。

优化：可以搞一个线程池来处理请求，但是高并发请求时，还是可能会导致各种排队和延时，因为没那么多线程来处理。





阻塞IO模型：

在调用ServerSocket.accept()方法时，会一直阻塞到客户端连接才会返回。

在调用InputStream.read()方法是阻塞的，它会一直等到数据到来是（或超时）才会返回；

两点缺点：当客户端多时，会创建大量的处理线程，且每个线程都要占用栈空间和一些CPU时间，且大量线程会带来频繁的上下文切换。

## NIO

JDK1.4中引入NIO，这是一种同步非阻塞IO，基于Reactor模型。

Java NIO采用了双向通道（channel）进行数据传输。

### NIO中的一些概念

NIO核心由三部分组成：缓冲区(Buffer)、通道(Channel)、选择器(Selector)

nio是面向块（缓冲区）编程，io是面向流编程。面向缓冲区编程：数据读写必须经过缓冲区。

Channel：NIO都是通过Channel进行数据读写的。

Buffer：缓冲区，将数据写入Buffer中，然后从Buffer中读取数据。

Selector：多路复用器，selector会不断轮询注册的channel，如果某个channel上发送了读写事件，selector就会将这些channel获取出来，我们通过SelectionKey获取有读写事件的channel，就可以进行IO操作了。一个Selector就使用一个线程，去轮询成千上万个channel，这就意味着你的服务端可以接入成千上万个客户端。

BIO的服务端接收到客户端的socket连接后，会创建一个socket和一个线程去跟客户端socket通信；而NIO的服务端接收到客户端socket连接后，只创建一个连接并注册到select，等selector轮询channel有读写事件时，再创建线程去进行IO操作。

原因是一个客户端不是时时刻刻都要发送请求的，没必要死耗着一个线程不放。所以NIO的优化思想是一个请求一个线程。只有某个客户端发送了一个请求时，才会启动一个线程来处理。

#### Buffer(缓存)

Buffer是一个抽象类，针对缓冲区封装的一个类，提供相应的方法来操作这个缓冲区。

子类：ByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, ShortBuffer

核心类：ByteBuffer和CharBuffer。

ByteBuffer有一个子类 MappedByteBuffer-》能够将文件直接映射到内存中，那么这样我们就可以像访问内存一样访问文件，非常方便

获取ByteBuffer：

static ByteBuffer allocate(int capacity); //分配一个新的字节缓冲区

static ByteBuffer allocateDirect(int capacity); //分配一个新的直接字节缓冲区

二者区别：

1、创建普通Buffer成本低，读写的效率不高

2、创建直接Buffer成本高，所以我们一般用在Buffer生存周期较长的时候使用

3、只有ByteBuffer才能够创建直接Buffer，其他的Buffer对象是不能够创建

4、如果创建了直接Buffer但是我又想要使用其他Buffer的功能，可以将ByteBuffer转换成其他Buffer。asIntBuffer()

Buffer的4个重要参数：

关系：0 < mark < postion < limit < capacity

capacity：buffer的大小/容量。不可以为负数，一旦创建就不能改变

position：当前读/写的位置。写操作到缓冲区时，position表示当前待写入的位置，position最大为capacity-1；从缓冲区读数据时，position表示从当前位置读取。

mark：标记索引，该索引能够用于下次读取或者写入，它只能够在0-position之间

limit：信息末尾的位置。limit后面的数据既不可读，也不可写。写操作，表示能够从buffer里写多少数据。

Buffer相关方法：

flip():将写模式切换为读模式， 将limit的值改为postion的值，同时将postion归0。特点: 就是为下一次数据的读取做好准备

clear(): 将读模式切换为写模式，将limit改为capacity的值，同时将postion归0。特点: 就是为下一次数据的写入做好准备

put(): 相对读取，向Buffer中存储数据

get(): 相对读取，从Buffer中获取数据

mark(): 设置标记位

reset(): 重置

hasRemaining(): 判断当前位置和limit之间是否还有元素可处理

绝对读取: get(index) 不会影响position的位置

相对读取: put() get() 会影响，每次读取一次，指针后移

#### Channel(通道)

3个常用方法：

read():将Channel中的数据读取到Buffer中

write():向Buffer中写入数据

map(mode, position, size):将channel中的数据全部或者部分映射到Buffer中。

如MappedByteBuffer mappBuffer = inChannel.map(MapMode.READ\_ONLY, 0, srcFile.length());

#### Selector(选择器)

一个组件，可以检测多个NIO Channel，看看读或写事件是否就绪。

多个Channel以事件的方式注册到同一个Selector，从而达到一个线程处理多个请求。

当你调用Selector的select()或者selectNow()方法它只会返回有数据读取的SelectableChannel的实例。

### 工作原理

每次客户端建立好连接，服务端就会创建一个对应的Channel并注册到selector上，然后selector轮询这些所有Channel，channel有请求过来的时候，它才创建一个工作线程去处理这个请求，处理后得到的结果也写入到该channel中，然后返回给客户端socket。

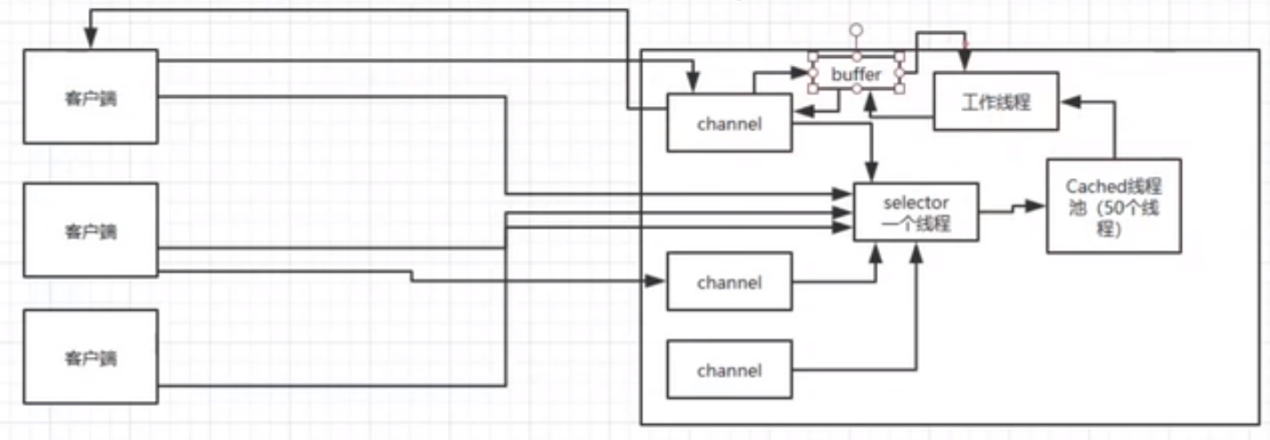
### 为什么NIO是同步非阻塞的呢？

问：为什么NIO是同步非阻塞的呢？

非阻塞理解：因为无论多少客户端都可以接入服务端，客户端接入并不会耗费一个线程只会创建一个连接然后注册到selector上，一个selector线程不断的轮询所有的socket连接，发现有事件了就会通知你，然后你就启动一个线程去处理该请求即可。但是这个处理过程中，你还是要先读取数据，处理，再返回的，这是个同步的过程。

工作线程向channel进行读写操作都是同步进行的：

从channel里读取数据，是工作线程自己去干这个事情，数据没读完，你就卡在那里；然后往channel里写数据，也是你自己去干的事情，数据没写完，你就卡在那里。



## AIO

AIO是基于Proactor模型的，就是异步非阻塞模型。

每个连接发送过来的请求，都会绑定一个buffer，然后通知操作系统去异步完成读，此时你的程序是会去干别的事情，等操作系统完成数据读取之后，就会回调你的接口，给你操作系统异步读完的数据。然后你对这个数据进行处理，接着把结果往回写。写的时候也是给操作系统一个buffer，让操作系统自己获取数据去完成写操作，写完之后再回来通知你。

工作线程的读数据和写数据：

读取数据时，工作线程会给操作系统提供一个空的buffer，然后就可以干别的事情了，也就是说把读数据的事情交给操作系统去干了。操作系统内核，将数据放入buffer后，就回调你的一个接口，告诉你以读取完毕。

写数据也是一样，把放了数据的buffer交给操作系统内核去处理，你就可以去干别的事情了，操作系统完成了数据的写之后，就会回调你，告诉你说，你交给我的数据，我都给你写回到客户端去了。

## BIO、NIO、AIO总结

1.BIO：

同步阻塞的IO模型，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务端就会启动一个线程去处理。

缺点：若连接不做任何事情会造成不必要的线程开销，可使用线程池机制改善。

BIO方式适用于连接数目比较小且固定的架构。

2.NIO：

描述：同步非阻塞的IO模型，服务器实现模式为一个请求一个线程，即客户端发送的连接都会注册到多路复用器上，多路复用器轮询到连接有IO请求时才启动一个线程进行处理。

NIO也是IO多路复用的基础，是解决高并发与大量连接、IO处理问题的有效方式。

NIO适用于连接数多且连接比较短（轻架构）的架构。比如聊天服务器，并发局限于应用中。

3.AIO：

描述：异步非阻塞的IO模型，服务器实现模式为一个有效请求一个线程，即客户端的IO请求都是由OS先完成了，再通知服务器应用区启动线程去处理。

AIO适用于连接数多且连接比较长（重架构）的架构。比如相册服务器，充分调用OS参与并发操作。

# IO模型

## 概念

阻塞、非阻塞（等待数据全部读取成功再返回，还是读取为空马上返回然后下次再读）

同步、异步（用户缓存主动去读取内核缓存，还是内核缓存读取磁盘成功后通知用户缓存）

NIO是同步非阻塞模型，也是IO多路复用基础。

Reactor模式基于同步IO，Proactor模式基于异步IO

IO模型：

IO的各种流是阻塞的，这意味着当一个线程调用read或write方法时，该线程被阻塞，直到有一些数据被读取，或数据完全写入，该线程在此期间不能做其他事情。

所有IO都分为两个阶段：等待就绪和操作。例如：读函数分为等待系统可读和真正的读；写函数分为等待网卡可写和真正的写。

等待就绪的阻塞是不使用CPU的，是在“空等”；真正的读写操作的阻塞是使用CPU的，而且这个过程非常快，属于memory copy，带宽通常在1GB/s级别以上，基本不耗时。

以socket.read()为例子：

BIO，若TCP RecvBuffer里没有数据，函数会一直阻塞，直到收到数据，返回读到的数据。

NIO，若TCP RecvBuffer有数据，就把数据从网卡读到内存，并且返回给用户；反之则直接返回0，永远不会阻塞。

AIO，不但等待就绪是非阻塞的，就连数据从网卡到内存的过程也是异步的。

4种IO模型：

1）同步阻塞IO(Blocking IO)：即传统的IO模型

2）同步非阻塞IO(Non-blocking IO)：默认创建的socket都是阻塞的，非阻塞IO要求socket被设置为Non-block。

3）IO多路复用(IO Nultiplexing)：即经典的Reactor设计模式，有时也称为异步阻塞IO，Java中的selector和Linux中的epoll都是这种模型

4）异步IO(Asynchronous IO)：即经典的Proactor设计模式，也称为异步非阻塞IO。

## IO多路复用

IO多路复用的场景是我们需要设计一个高性能的网络服务器。该网络服务器能同时跟多个客户端连接，并且能处理这些客户端传上来的请求。

文件描述符，简称fd

如何设计一个IO多路复用的网络服务器？

设计1：我们可以写一个多线程的程序，每个传上来的请求都是一个线程。但是多线程会存在上下文切换。

设计2：如何用单线程来处理大量客户端连接请求？引出select/poll/epoll

## select、poll、epoll介绍

select、poll、epoll都是IO多路复用中的模型，一般用户程序基于这个去开发自己的IO复用模型。如NIO的非阻塞模型，就是采用了IO多路复用的方式，是基于epoll实现的。

1、select通过数据来存储。通过轮询数组，才能感知哪个socket来了数据。缺点是数组长度只能是1024，且需要不断在内核空间和用户空间之间拷贝数组。

2、poll基于链表存储。它将用户传入的数组拷贝到内核空间，然后查询每个fd对应的设备状态。本质上跟select没有区别，轮询链表，且需要拷贝数据，但它没有最大连接数的限制。

1和2两种方法适合在请求总数比较少，并且活跃请求数较多的情况。1和2都需要在内核空间和用户空间之间不断拷贝数据。

3、epoll基于红黑树存储。它在内核空间开辟该数据结构，树节点中存放的是一个socket描述符以及用户程序感兴趣的事件类型。同时epoll还会维护一个链表，用于存储已经就绪的socket描述符节点。

select模型：

NIO最主要的就是实现了对异步操作的支持。其中一种通过把一个套接字通道(SocketChannel)注册到一个选择器(Selector)中，不时调用选择器的select方法就能返回满足的选择键(SelectionKey)，键中包含了SOCKET事件信息，这就是select模型。

select和epoll区别？

1）select有最大并发数限制，默认最大文件句柄数1024，可修改。epoll没有最大文件句柄数限制，仅受系统中进程能打开的最大文件句柄数限制。

2）select每次都要线性扫描fd\_set集合。epoll只在集合不为空才轮询。

3）select存在内核空间和用户空间的内存拷贝问题。epoll中减少内存拷贝，map将用户空间的一块地址和内核空间的一块地址同时映射到相同的一块物理内存地址。

问：Epoll导致的selector空轮询？

描述：Java NIO Epoll 会导致 Selector 空轮询，最终导致 CPU 100%。

官方声称在JDK1.6版本的update18修复了该问题，但是直到JDK1.7版本该问题仍旧存在，只不过该BUG发生概率降低了一些而已，它并没有得到根本性解决。

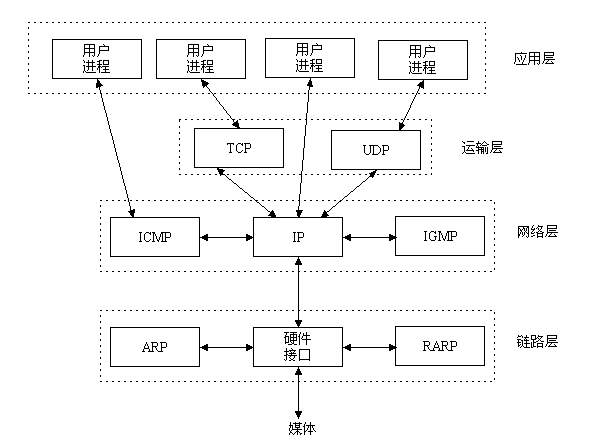
Netty的解决方案：对 Selector 的 select 操作周期进行统计，每完成一次空的 select 操作进行一次计数，若在某个周期内连续发生 N 次空轮询，则判断触发了 Epoll 死循环 Bug。然后，Netty 重建 Selector 来解决。判断是否是其他线程发起的重建请求，若不是则将原 SocketChannel 从旧的 Selector 上取消注册，然后重新注册到新的 Selector 上，最后将原来的 Selector 关闭。

# TCP/IP协议簇

TCP/IP（Transaction Control Protocol/Internet Protocol）传输控制协议/网间协议，是一个工业标准的协议集，它是为广域网（WANs）设计的。

UDP（User Data Protocol，用户数据报协议）是与TCP相对应的协议，它属于TCP/IP协议簇中的一种。

TCP/IP协议族包括运输层、网络层、链路层。



问：网络中的进程如何通信？

在本地可以通过进程PID唯一标识一个进程，但是在网络中不行。

其实TCP/IP协议簇已经帮我们解决了这个问题：“网络层的IP地址”可以唯一标识网络中的主机，而传输层的“协议+端口”可以唯一标识主机中的应用程序（进程），这样利用三元组（IP地址，协议，端口）就可以标识网络的进程了，网络中的进程通信就可以利用这个标志与其它进程进行交互。

使用TCP/IP协议的应用程序通常采用应用编程接口：套接字(Socket)，来实现网路进程之间的通信。就目前而言，几乎所有的应用程序都是采用socket，而现在又是网络时代，网络中进程通信是无处不在，这就是我为什么说“一切皆socket”。

# Socket介绍

进程间通信通过socket来进行。

先从服务器端说起。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。



## Socket工作原理

各层之间的协议：

应用层：http协议

传输层：tcp协议，socket属于传输层的这么一个变成规范

网络层：ip协议

数据链路层：以太网协议

Socket工作原理：

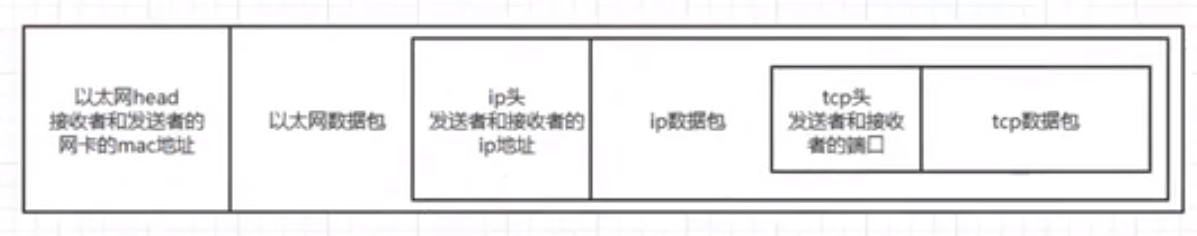
Socket在应用层和传输层之间的一个抽象层，它把 TCP/IP 层复杂的操作抽象为几个简单的接口，供应用层调用实现进程在网络中的通信。

说白了，socket就是在传输层把tcp/ip协议给封装了一下，我们程序员一般都是面向socket来编程的。

大体来说这个步骤，就是我们创建一个ServerSocket来无限等待别人来连接你，然后某个机器要跟你连接，就在本地创建一个socket去连接你，然后与服务器建立连接后，ServerSocket也会创建一个socket。通过客户端的socket与服务端的socket进行通信我给你写数据，你读数据，你给我写数据，我读数据，就这个过程。

其它底层，如建立连接和释放连接，都是基于tcp三次握手和四次挥手的规范来实现的。数据传输也是基于tcp协议的，把数据封装在tcp数据包，里面包括有tcp报头、端口号这些，然后封装在ip数据包，最后封装在以太网数据包进行传输。



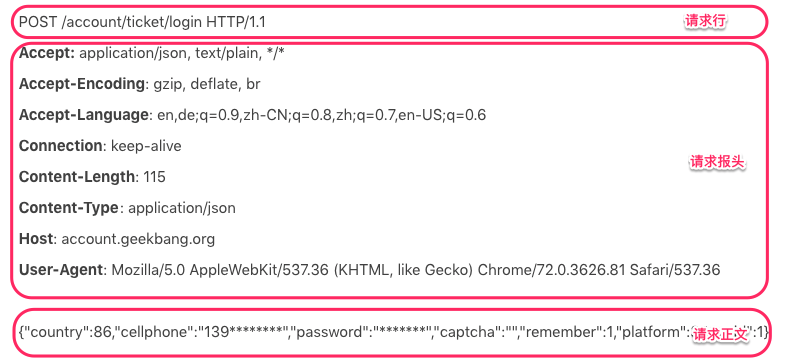


# Http介绍

## http数据包

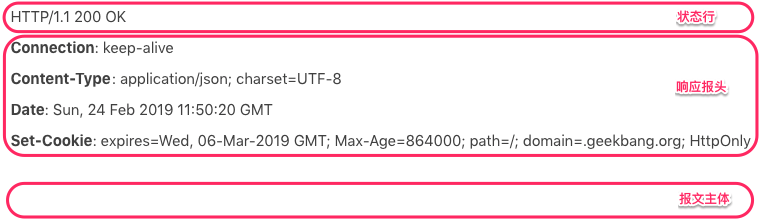
当http请求数据到达Tomcat后，Tomcat会把Http请求数据字节流解析成一个Request对象，这个Request对象封装了Http所有的请求信息。接着Tomcat把这个Request对象交给Web应用去处理，处理完后得到一个Response对象，Tomcat会把这个Response对象转换成Http格式的响应数据并发送给浏览器。

HTTP请求数据：请求行、请求头、请求正文



登录请求示例图

HTTP响应数据：状态行、响应头、报文主体



登录请求的响应示例图

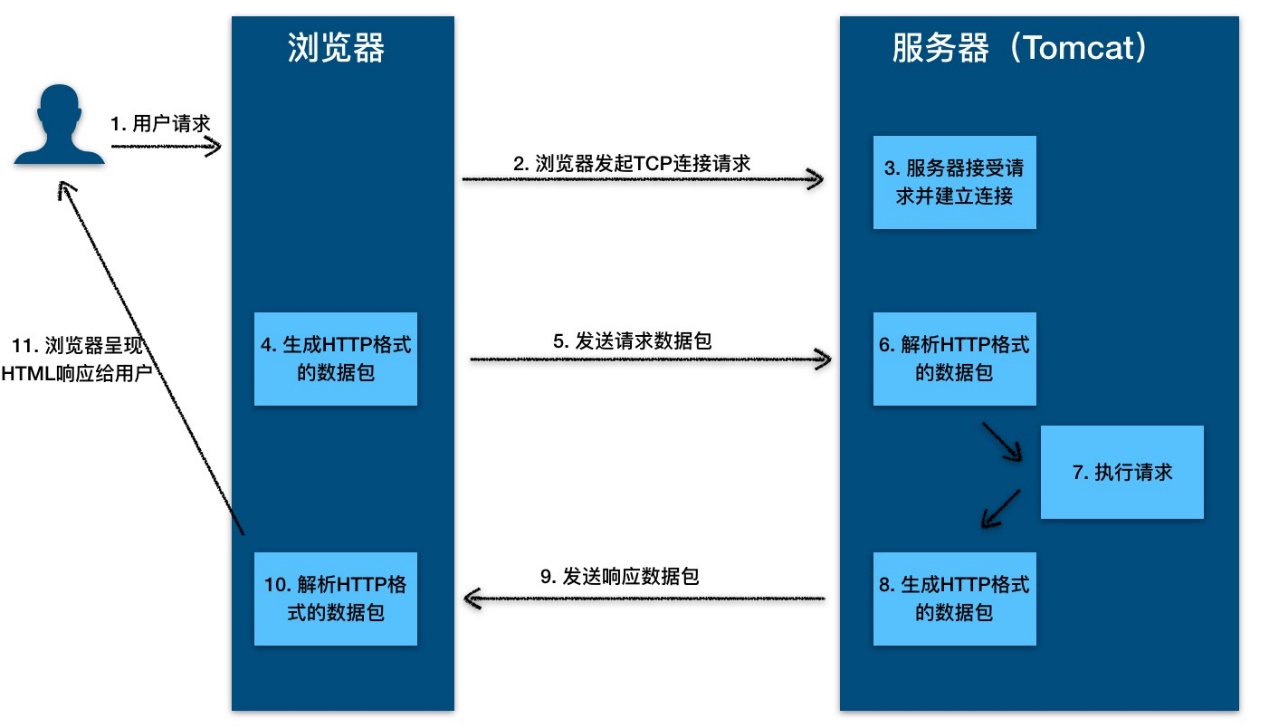
## http请求过程

浏览器从远程http服务器获取一个html文本过程中主要做两件事：

-1）与服务器建立Socket连接

-2）生成请求数据并通过Socket发送出去

-3）解析服务器发送过来的响应数据



说明：

-1）用户通过浏览器访问某网址或点击某a标签；

-2）浏览器向服务器发送tcp连接请求；

-3）服务器接收浏览器的连接请求，并经过tcp三次握手后建立连接；

-4）浏览器将请求数据打包成http协议格式的数据包；

-5）浏览器将该数据包推入网络，数据包经过网络传输，最终达到服务端程序；

-6）服务端程序拿到这个数据包后，同样以http协议格式解包；

-7）服务器通过解包数据执行请求，生成响应结果；

-8）服务端将响应结果（html文本、图片）打包成http协议格式的数据包；

-9）服务器将响应数据包推入网络，数据包经过网络传输最终达到浏览器；

-10）浏览器拿到响应数据包后，以http协议的格式解包，然后解析数据，假设这里的数据是html；

-11）浏览器将html文件展现在页面上。

建立tcp连接说明：

在http工作开始之前web浏览器首先通过网络与web服务器建立连接。

连接是通过tcp实现，tcp协议与ip协议共同构建Internet，即著名的tcp/ip协议族，因此Internet又被称为tcp/ip网络。

http是比tcp更高层的应用层协议，根据规则，只有低层次协议建立之后才能进行更高层协议的连接，因此首先要建立tcp连接。一般tcp连接的端口是80。

Tomcat和Jetty作为http服务器，在这个过程中都做了什么？

接收并建立连接、解析请求数据、处理请求、发送响应

附：若有成千上万的浏览器请求同一个http服务器，因此Tomcat和Jetty为了提高服务的能力和并发度，往往会将自己要做的几个事情并行化，具体使用多线程技术。

## Http中间人攻击

参考https的中间人攻击。

有做过修改host文件来访问谷歌，这种就是通过中间人来访问谷歌的。

## http2.0介绍

http2.0特点：

1、多路复用技术。该技术可以只通过一个tcp连接就可以传输所有的请求数据。

2、新的编码机制。所有传输的数据都会被分割，并采用二进制格式编码。(以前httpb版本中通过文本的方式传输数据)

3、在http/2中有两个重要的概念：帧frame和流stream。帧代表着最小的数据单位，每个帧会标识出该帧属于哪个流，流也就是多个帧组成的数据流。

http2特点：https://http2.akamai.com/demo

# https介绍

## https流程

https流程：

主要包括证书验证(非对称加密)和数据传输(对称加密)两大步骤。

--证书验证阶段：

-1）浏览器发起https请求，发送握手消息；

-2）服务端返回ssl证书、对称加密算法和公钥给客户端；

-3）客户端验证证书是否合法，若不合法则提示非安全证书。

--数据传输阶段：

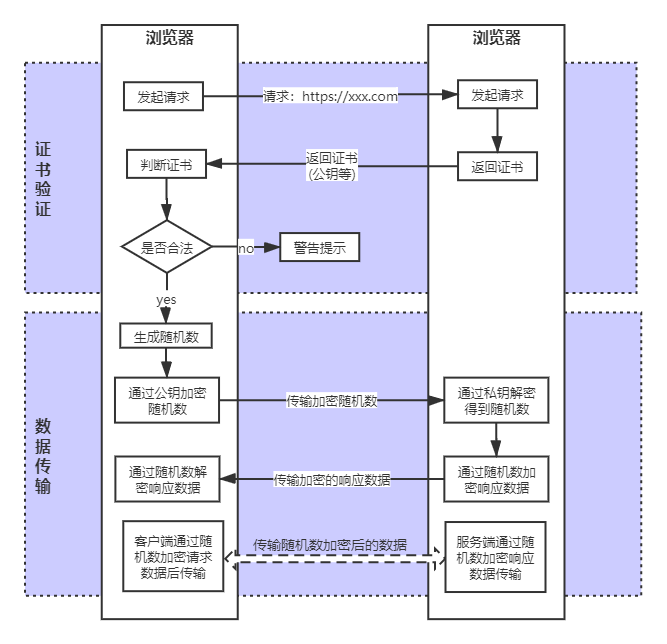
-1）当证书验证合法后，在本地生成随机数，并将随机数用证书里加密算法的公钥进行加密后发送给服务端；

-2）服务端通过私钥进行解密后得到随机数，并用此随机数作为密钥采用对称加密算法加密一段握手消息发给客户端；

-3）客户端收到消息后解密成功，则握手结束。

--发送请求：

客户端将请求参数通过随机数加密后传输，服务器使用随机数解密得到请求参数。



公钥、私钥和随机数的作用：

公钥和私钥的作用：客户端使用公钥对随机数加密传给服务端，服务端通过私钥解密得到随机数。

随机数作用：用于数据传输的对称加密算法的加密。

为什么数据传输是用对称加密？

1、非对称加密的加密解密效率非常低，而服务端与客户端之间存在大量的交互，非对称加密的效率无法接受；

2、在https的场景只有服务端保存了私钥，一对公私钥只能实现单向的加解密，所以https中内容传输加密采取的是对称加密，而不是非对称加密。

本地随机数被窃取怎么办？

描述：证书验证是采用非对称加密实现，但是传输过程是采用对称加密，而其中对称加密算法中重要的随机数是由本地生成并且存储于本地的，HTTPS如何保证随机数不会被窃取？

其实HTTPS并不包含对随机数的安全保证，HTTPS保证的只是传输过程安全，而随机数存储于本地，本地的安全属于另一安全范畴，应对的措施有安装杀毒软件、反木马、浏览器升级修复漏洞等。

问：如果我篡改了公钥呢？

公钥只是用来加密随机数的，篡改了会导致服务端拿不到随机数。

## https证书

https证书包含什么信息？

签发机构、用户信息、公钥、域名、有效期、指纹等信息。

浏览器如何验证ssl证书的合法性？

证书来源是否合法；

域名、有效期等信息是否正确；

证书是否被篡改，需要与ca服务器进行校验。

如何避免证书的冒用？

描述：证书是公开的，若要发起中间人攻击，我在官网上下载一份证书作为我的服务器证书，那客户端肯定会认同这个证书是合法的。

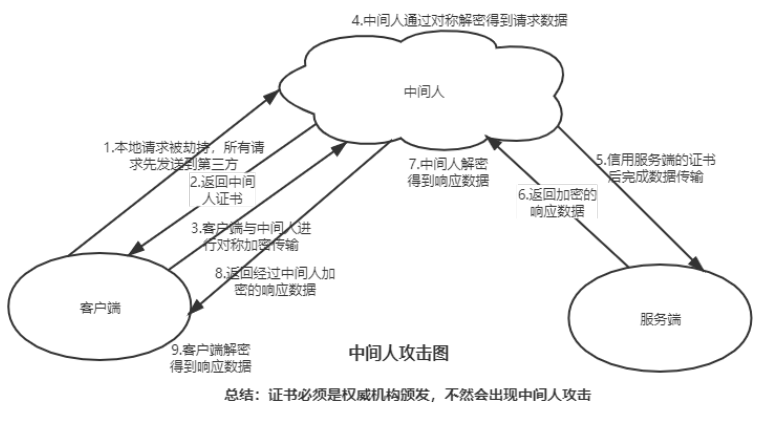
答案：证书来源不合法，且证书上的域名和网址上的域名不一致，客户端校验不通过。

证书的合法性依据是什么？

证书由ca认证机构颁发，不是随便一个机构都有资格颁发证书的。权威机构需要对其颁发的证书进行信用背书，我们能认为证书是合法的。但不同等级的权威机构对审核的要求也不一样，因此证书也分免费和付费的。

为什么需要ca认证机构颁发证书？

假设不存在认证机构，任何人都可以制作证书，这带来的安全风险就是经典的“中间人攻击”问题。



说明：

-1）本地请求被劫持（如DNS劫持等），所有请求均被发送到中间人的服务器；

-2）中间人服务器返回中间人自己的证书；

-3）客户端创建随机数，通过中间人证书的公钥对随机数加密后传送给中间人，然后凭随机数对传输内容进行加密传输；

-4）中间人因为拥有客户端的随机数，可以用过对称加密算法进行内容解密；

-5）中间人以客户端请求内容再向正规网站发起请求

-6）因为中间人与服务器的通信过程是合法的，正规网站通过建立的安全通道返回加密后的数据。

-7）中间人凭借与正规网站建立的对称加密算法对内容进行解密。

-8）中间人通过与客户端建立的对称加密算法对正规内容返回的数据进行加密传输。

-9）客户端通过与中间人建立的对称加密算法对返回结果数据进行解密。

## https流程

证书必须要权威机构颁发，不然会出现跟http一样的中间人攻击。