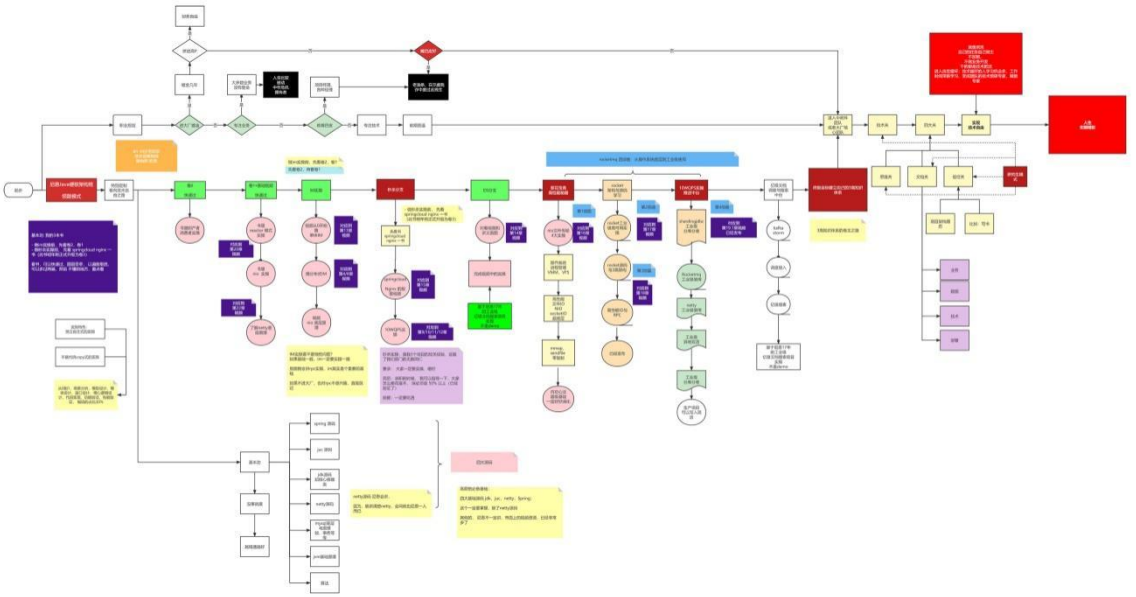
牛逼的职业发展之路

40 岁老架构尼恩用一张图揭秘：Java 工程师的高端职业发展路径，走向食物链顶端的之路

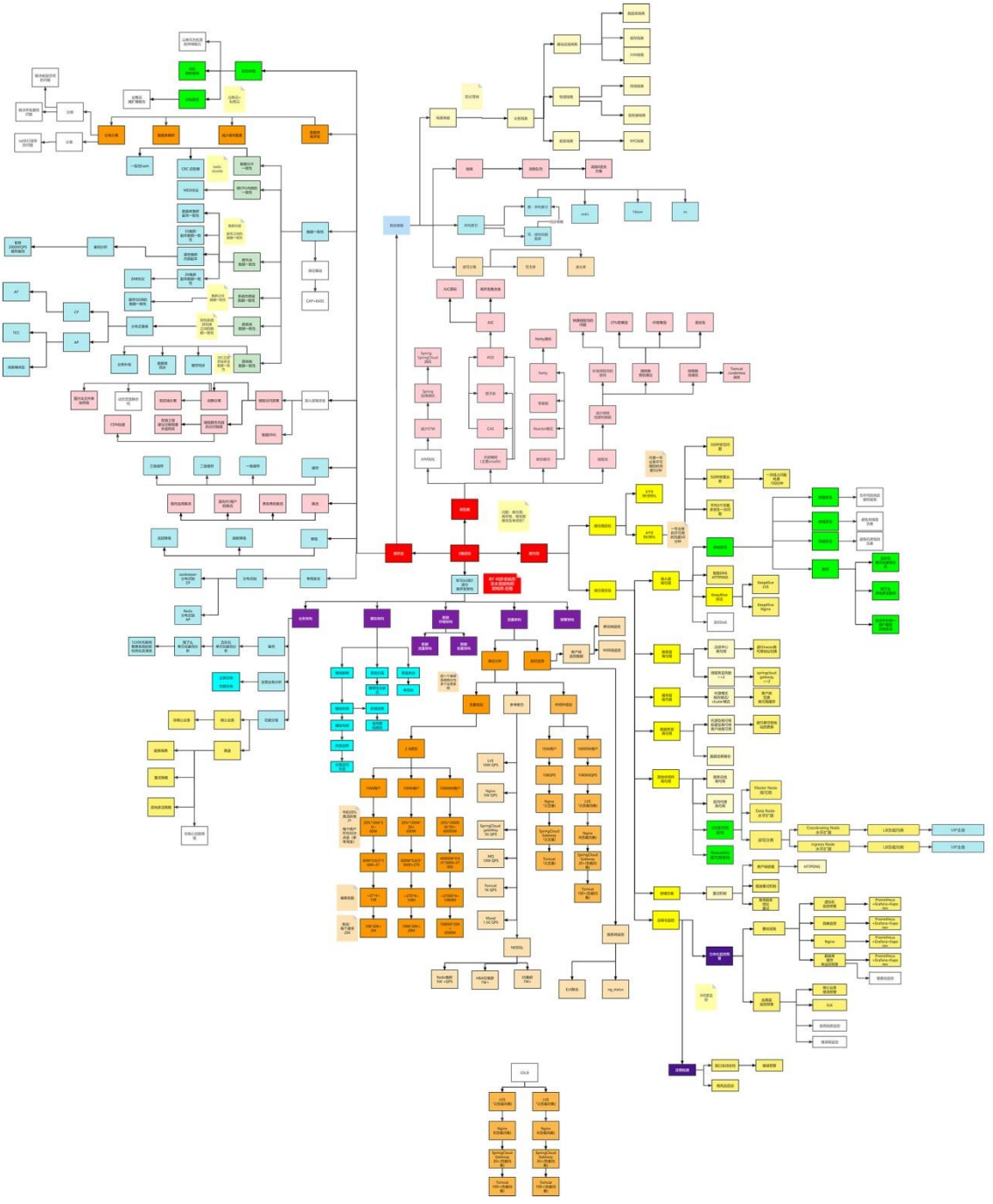
链接：https://www.processon.com/view/link/618a2b62e0b34d73f7eb3cd7



史上最全：价值10W的架构师知识图谱

此图梳理于尼恩的多个 3 高生产项目：多个亿级人民币的大型 SAAS 平台和智慧城市项目

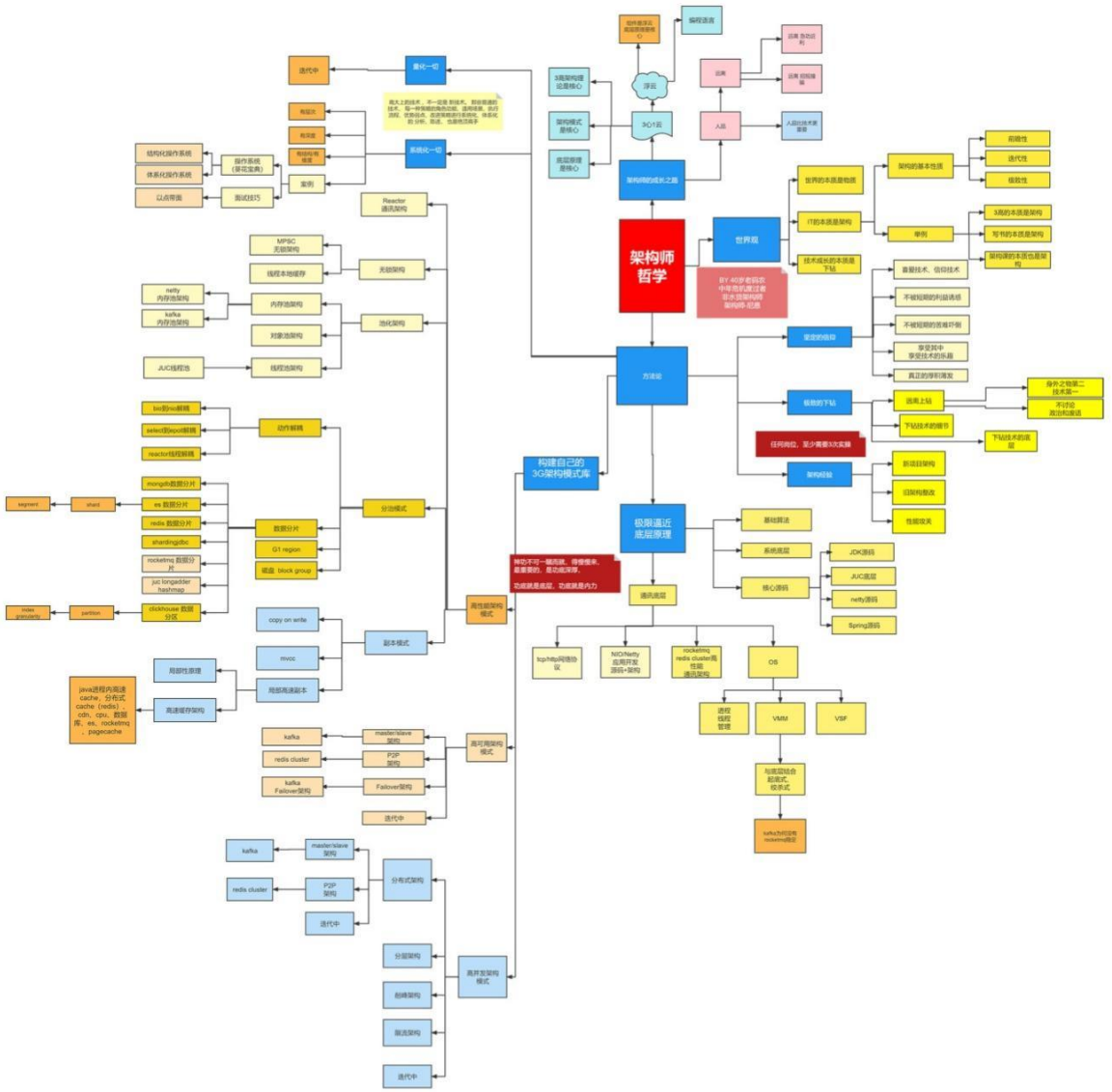
链接：https://www.processon.com/view/link/60fb94216376897l9d246739



牛逼的架构师哲学

40 岁老架构师尼恩对自己的 20 年的开发、架构经验总结

链接：https://www.processon.com/view/link/616f801963768961e9d9aec8



牛逼的3高架构知识宇宙

尼恩 3 高架构知识宇宙，帮助大家穿透 3 高架构，走向技术自由，远离中年危机

链接：https://www.processon.com/view/link/635097d2e0b34d40be778ab4

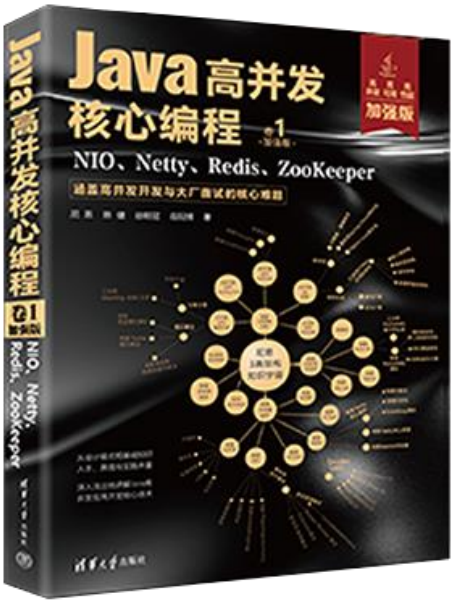
尼恩Java高并发三部曲（卷1加强版）

老版本：《Java 高并发核心编程 卷 1：NIO、Netty、Redis、ZooKeeper》（已经过时，不建议购买）

新版本：《Java 高并发核心编程 卷 1 **加强版**：NIO、Netty、Redis、ZooKeeper》

- 由浅入深地剖析了高并发 IO 的底层原理。
- 图文并茂的介绍了 TCP、HTTP、WebSocket 协议的核心原理。
- 细致深入地揭秘了 Reactor 高性能模式。
- 全面介绍了 Netty 框架，并完成单体 IM、分布式 IM 的实战设计。
- 详尽地介绍了 ZooKeeper、Redis 的使用，以帮助提升高并发、可扩展能力

详情：https://www.cnblogs.com/crazymakercircle/p/16868827.html



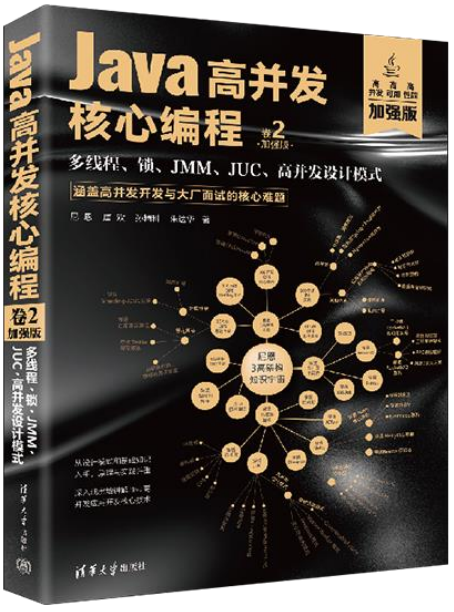
尼恩Java高并发三部曲（卷2加强版）

老版本：《Java 高并发核心编程 卷 2：多线程、锁、JMM、JUC、高并发设计模式》（已经过时，不建议购买）

新版本：《Java 高并发核心编程 卷 2 **加强版**：多线程、锁、JMM、JUC、高并发设计模式》

- 由浅入深地剖析了 Java 多线程、线程池的底层原理。
- 总结了 IO 密集型、CPU 密集型线程池的线程数预估算法。
- 图文并茂的介绍了 Java 内置锁、JUC 显式锁的核心原理。
- 细致深入地揭秘了 JMM 内存模型。
- 全面介绍了 JUC 框架的设计模式与核心原理，并完成其高核心组件的实战介绍。
- 详尽地介绍了高并发设计模式的使用，以帮助提升高并发、可扩展能力

详情参阅：https://www.cnblogs.com/crazymakercircle/p/16868827.html



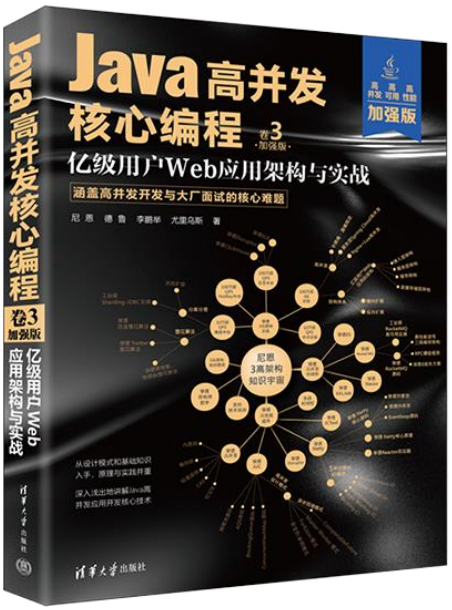
尼恩Java高并发三部曲（卷3加强版）

老版本：《SpringCloud Nginx 高并发核心编程》（已经过时，不建议购买）

新版本：《Java 高并发核心编程 卷 3 **加强版**：亿级用户 Web 应用架构与实战》

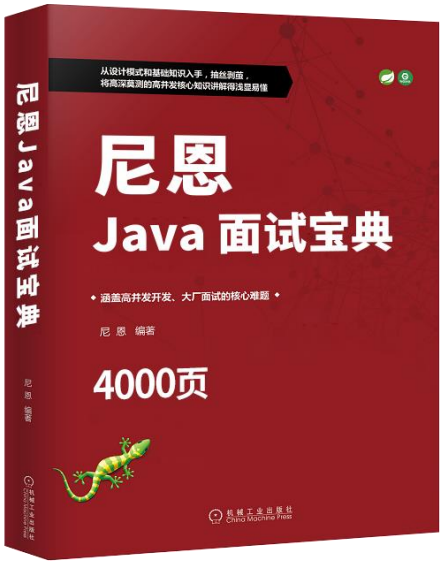
- 在当今的面试场景中，3 高知识是大家面试必备的核心知识，本书基于亿级用户 3 高 Web 应用的架构分析理论，为大家对 3 高架构系统做一个系统化和清晰化的介绍。
- 从 Java 静态代理、动态代理模式入手，抽丝剥茧地解读了 Spring Cloud 全家桶中 RPC 核心原理和执行过程，这是高级 Java 工程师面试必备的基础知识。
- 从Reactor 反应器模式入手，抽丝剥茧地解读了Nginx 核心思想和各配置项的底层知识和原理，这是高级 Java 工程师、架构师面试必备的基础知识。
- 从观察者模式入手，抽丝剥茧地解读了 RxJava、Hystrix 的核心思想和使用方法，这也是高级 Java 工程师、架构师面试必备的基础知识。

详情：https://www.cnblogs.com/crazymakercircle/p/16868827.html



尼恩Java面试宝典

40 个专题（卷王专供+ 史上最全 + 2023 面试必备）
详情：https://www.cnblogs.com/crazymakercircle/p/13917138.html



专题01：JVM面试题（卷王专供 + 史上最全 + 2023面试必备）-V20-from-尼恩Java面试宝典.pdf
专题02：Java算法面试题（卷王专供 + 史上最全 + 2023面试必备）-V2 -from-Java面试红宝书-release.pdf
专题03：Java基础面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书 -release.pdf
专题04：架构设计面试题（卷王专供+ 史上最全 + 2023面试必备）-V24-from-Java面试红宝书-release.pdf
专题05：Spring面试题_专题06：SpringMVC_专题07：Tomcat面试题（卷王专供+ 史上最全 + 2023面试必备）-V3-from-尼恩面试宝典-release.pdf
专题08：SpringBoot面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题09：网络协议面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题10：TCPIP协议（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题11：JUC并发包与容器类（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题12：设计模式面试题 （卷王专供+ 史上最全 + 2023面试必备）-V14-from-Java面试红宝书.pdf
专题13：死锁面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题14：Redis 面试题 （卷王专供+ 史上最全 + 2023面试必备）-V22-from-Java面试红宝书-release.pdf
专题15：分布式锁 面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题16：Zookeeper 面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题17：分布式事务面试题（卷王专供 + 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题18：一致性协议 （卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题19：Zab协议（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题20：Paxos 协议（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题21：raft 协议（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题22：Linux面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题23：Mysql 面试题（卷王专供+ 史上最全 + 2023面试必备）-V29-from-尼恩Java面试宝典-release.pdf
专题24：SpringCloud 面试题（卷王专供+ 史上最全 + 2023面试必备）-V12-from-Java面试红宝书-release.pdf
专题25：Netty 面试题 （卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题26：消息队列面试题：RabbitMQ、Kafka、RocketMQ（卷王专供+ 史上最全 + 2023面试必备）-V10-from-Java面试红宝书-release.pdf
专题27：内存泄漏 内存溢出（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题28：JVM 内存溢出 实战（卷王专供+ 史上最全 + 2023面试必备）-V17-from-Java面试红宝书-release.pdf
专题29：多线程面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题30：HR面试题：过五关斩六将后，小心阴沟翻船！（史上最全、避坑宝典）-V2-from-Java面试红宝书-release.pdf
专题31：Hash连环炮面试题（卷王专供+ 史上最全 + 2023面试必备）-V16-from-Java面试红宝书-release.pdf
专题32：大厂面试的基本流程和面试准备（阿里、腾讯、网易、京东、头条.....）-V2-from-Java面试红宝书-release.pdf
专题33：BST、AVL、RBT红黑树、三大核心数据结构（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题34：Elasticsearch面试题（卷王专供+ 史上最全 + 2023面试必备）-V3-from-Java面试红宝书-release.pdf
专题35：Mybatis面试题（卷王专供+ 史上最全 + 2023面试必备）-V3-from-尼恩Java面试宝典-release.pdf
专题36：Dubbo面试题（卷王专供+ 史上最全 + 2023面试必备）-V21-from-尼恩Java面试宝典-release.pdf
专题37：Docker面试题（卷王专供+ 史上最全 + 2023面试必备）-V26-from-尼恩Java面试宝典-release.pdf
专题38：K8S面试题（卷王专供+ 史上最全 + 2023面试必备）-V26-from-尼恩Java面试宝典-release.pdf
专题39：Nginx面试题（卷王专供+ 史上最全 + 2023面试必备）-V27-from-尼恩Java面试宝典-release.pdf
专题40：操作系统面试题（卷王专供+ 史上最全 + 2023面试必备）-V28-from-尼恩Java面试宝典-release.pdf

专题9：网络协议面试题（史上最全、定期更新）

本文版本说明：V2

> 此文的格式，由markdown 通过程序转成而来，由于很多表格，没有来的及调整，出现一个格式问题，尼恩在此给大家道歉啦。
>
> 由于社群很多小伙伴，在面试，不断的交流最新的面试难题，所以，《Java面试红宝书》， 后面会不断升级，迭代。
>
> 本专题，作为 《Java面试红宝书》专题之一， 《Java面试红宝书》一共**30个面试专题，后续还会增加**

《Java面试红宝书》升级的规划为：

后续基本上，**每一个月，都会发布一次**，最新版本，可以扫描扫架构师尼恩微信， 发送 “领取电子书” 获取。

尼恩的微信二维码在哪里呢？ 请参见文末

面试问题交流说明：

如果遇到面试难题，或者职业发展问题， 或者中年危机问题，都可以来 疯狂创客圈社群交流，

加入交流群，加尼恩微信即可，

HTTP协议

当你用浏览器打开一个链接的时候，计算机做了哪些工作步骤。

（1）、解析域名。

（2）、发起TCP的3次握手。

（3）、建立TCP请求后发起HTTP请求。

（4）、服务器相应HTTP请求。

（5）、浏览器得到HTML代码，进行解析和处理JSON数据，并请求HTML代码中的静态资源（JS、CSS、图片等）。

（6）、浏览器对页面进行渲染。

HTTP有哪些method。

★ GET：获取资源。

★ POST：表单提交。

★ HEAD：获取报头信息，HEAD 方法与 GET 方法类似，但并不会返回响应主体。

★ PUT 与PATCH：更新资源，PUT 对后台来说 PUT 方法的参数是一个完整的资源对象，它包含了对象的所有字段，PATCH 对后台来说 PATCH 方法的参数只包含我们需要修改的资源对象的字段。

★ DELETE：删除资源。

★ OPTIONS：获取目标资源所支持的通信选项,使用 OPTIONS 方法对服务器发起请求,以检测服务器支持哪些 HTTP 方法。

HTTPS的加密方式是什么，讲讲整个加密解密流程。

加密方式： 1）、对称密码算法：指加密和解密使用相同的密钥，速度高，可加密内容较大，用来加密会话过程中的消息。典型算法DES、AES、RC5、IDEA（分组加密）RC4。

2）、非对称密码算法：又称为公钥加密算法，是指加密和解密使用不同的密钥，加密速度较慢，但能提供更好的身份认证技术，用来加密对称加密的密钥（公开的密钥用于加密，私有的密钥用于解密）典型的算法RSA、DSA、DH。

3）、散列算法：将文件内容通过此算法加密变成固定长度的值（散列值），这个过程可以使用密钥也可以不使用。这种散列变化是不可逆的，也就是说不能从散列值编程原文，因此散列变化通道常用语验证原文是否被篡改。典型的算法MD5、SHA、BASE64、CRC等。

Http与Https的区别：

HTTP 的URL 以http:// 开头，而HTTPS 的URL 以https:// 开头

HTTP 是不安全的，而 HTTPS 是安全的

HTTP 标准端口是80 ，而 HTTPS 的标准端口是443

在OSI 网络模型中，HTTP工作于应用层，而HTTPS 的安全传输机制工作在传输层

HTTP 无法加密，而HTTPS 对传输的数据进行加密

HTTP无需证书，而HTTPS 需要CA机构颁发的SSL证书

什么是Http协议无状态协议?怎么解决Http协议无状态协议?

无状态协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息

也就是说，当客户端一次HTTP请求完成以后，客户端再发送一次HTTP请求，HTTP并不知道当前客户端是一个”老用户“。

可以使用Cookie来解决无状态的问题，Cookie就相当于一个通行证，第一次访问的时候给客户端发送一个Cookie，当客户端再次来的时候，拿着Cookie(通行证)，那么服务器就知道这个是”老用户“

URI和URL的区别

URI，是uniform resource identifier，统一资源标识符，用来唯一的标识一个资源。

Web上可用的每种资源如HTML文档、图像、视频片段、程序等都是一个来URI来定位的

URI一般由三部组成：

①访问资源的命名机制

②存放资源的主机名

③资源自身的名称，由路径表示，着重强调于资源。

URL是uniform resource locator，统一资源定位器，它是一种具体的URI，即URL可以用来标识一个资源，而且还指明了如何locate这个资源。

URL是Internet上用来描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上，特别是著名的Mosaic。

采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。URL一般由三部组成：

①协议(或称为服务方式)

②存有该资源的主机IP地址(有时也包括端口号)

③主机资源的具体地址。如目录和文件名等

常见的HTTP相应状态码

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优化方案

我下面就简要概括一下：

TCP复用：TCP连接复用是将多个客户端的HTTP请求复用到一个服务器端TCP连接上，而HTTP复用则是一个客户端的多个HTTP请求通过一个TCP连接进行处理。前者是负载均衡设备的独特功能；而后者是HTTP 1.1协议所支持的新功能

内容缓存：将经常用到的内容进行缓存起来，那么客户端就可以直接在内存中获取相应的数据了。

压缩：将文本数据进行压缩，减少带宽

SSL加速（SSL Acceleration）：使用SSL协议对HTTP协议进行加密，在通道内加密并加速

TCP缓冲：通过采用TCP缓冲技术，可以提高服务器端响应时间和处理效率，减少由于通信链路问题给服务器造成的连接负担。

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传输数据是通过URL请求，以field（字段）= value的形式，置于URL后，并用"?"连接，多个请求数据间用"&"连接，如http://127.0.0.1/Test/login.action?name=admin&password=admin，这个过程用户是可见的；

post传输数据通过Http的post机制，将字段与对应值封存在请求实体中发送给服务器，这个过程对用户是不可见的；

区别三：

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；

区别四：

get是不安全的，因为URL是可见的，可能会泄露私密信息，如密码等；

post较get安全性较高；

区别五：

get方式只能支持ASCII字符，向服务器传的中文字符可能会乱码。

post支持标准字符集，可以正确传递中文字符。

HTTP请求报文与响应报文格式

请求报文包含三部分：

a、请求行：包含请求方法、URI、HTTP版本信息

b、请求首部字段

c、请求内容实体

响应报文包含三部分：

a、状态行：包含HTTP版本、状态码、状态码的原因短语

b、响应首部字段

c、响应内容实体

Session和cookie的区别。

（1）、Cookie保存在客户端，未设置存储时间的Cookie，关闭浏览器会话Cookie就会被删除；设置了存储时间的Cookie保存在用户设备的磁盘中知道过期，同时Cookie在客户端所以可以伪造，不是十分安全，敏感数据不易保存。Session保存在服务器端，存储在IIS的进程开辟的内存中，而Session过多会消耗服务器资源，所以尽量少使用Session。

（2）、Session是服务器用来跟踪用户的一种手段，每个Session都有一个唯一标识：session ID。当服务端生成一个Session时就会向客户端发送一个Cookie保存到客户端，这个Cookie保存的是Session的SessionId这样才能保证客户端发起请求后，用户能够与服务器端成千上万的Session进行匹配，同时也保证了不同页面之间传值的正确性.

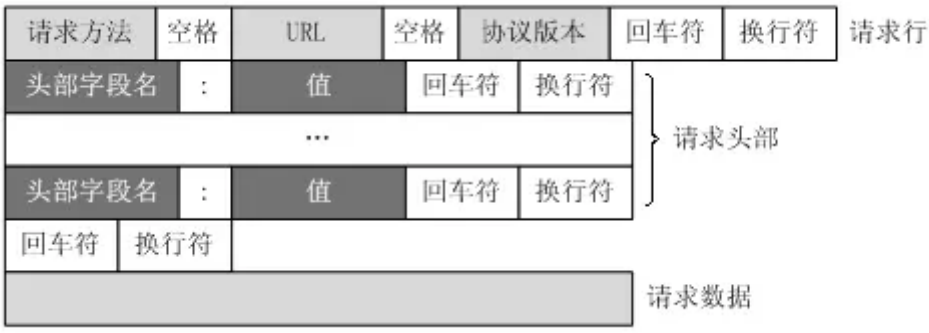
（3）、存储数据类型不同：Session能够存储任意的JAVA对象，Cookie只能存储String类型的对象。

（4）、长于10K的数据，不要用到Cookies。

http的请求报文是什么样的?

请求报文有4部分组成:

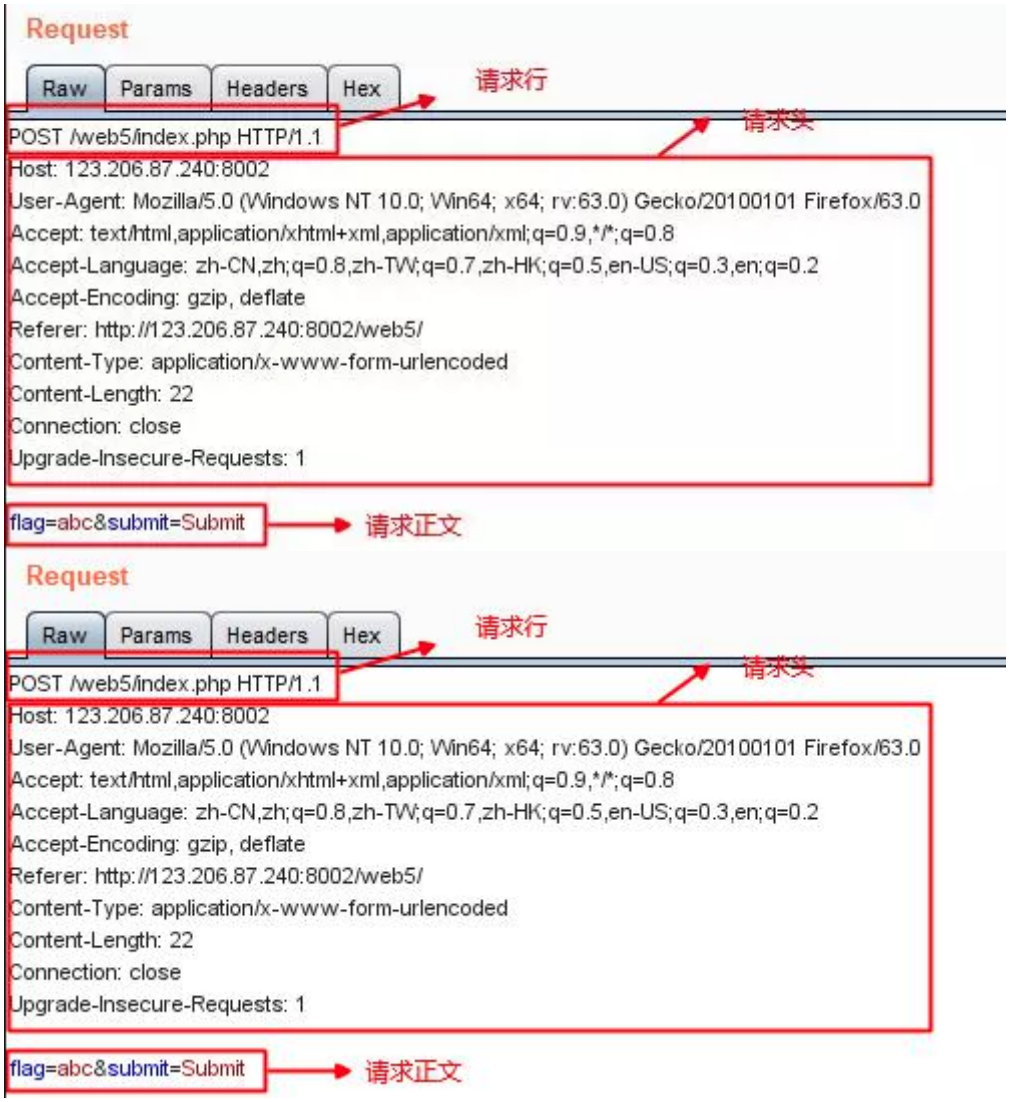
- 请求行
- 请求头部
- 空行
- 请求体



- 请求行包括：请求方法字段、URL字段、HTTP协议版本字段。它们用空格分隔。例如，GET /index.html HTTP/1.1。
- 请求头部:请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔

1. User-Agent：产生请求的浏览器类型。
2. Accept：客户端可识别的内容类型列表。
3. Host：请求的主机名，允许多个域名同处一个IP地址，即虚拟主机。

- 请求体: post put等请求携带的数据



http的响应报文是什么样的?

请求报文有4部分组成:

- 响应行
- 响应头
- 空行
- 响应体

- 响应行： 由协议版本，状态码和状态码的原因短语组成，例如`HTTP/1.1 200 OK`。
- 响应头：响应部首组成
- 响应体：服务器响应的数据

聊一聊HTTP的部首有哪些?

> 内容很多，重点看标『✨』内容

通用首部字段（General Header Fields）：请求报文和响应报文两方都会使用的首部

- Cache-Control 控制缓存 ✨
- Connection 连接管理、逐条首部 ✨
- Upgrade 升级为其他协议
- via 代理服务器的相关信息
- Wraning 错误和警告通知
- Transfor-Encoding 报文主体的传输编码格式 ✨
- Trailer 报文末端的首部一览
- Pragma 报文指令
- Date 创建报文的日期

请求首部字段（Reauest Header Fields）:客户端向服务器发送请求的报文时使用的首部

- Accept 客户端或者代理能够处理的媒体类型 ✨
- Accept-Encoding 优先可处理的编码格式
- Accept-Language 优先可处理的自然语言
- Accept-Charset 优先可以处理的字符集
- If-Match 比较实体标记（ETage） ✨
- If-None-Match 比较实体标记（ETage）与 If-Match相反 ✨
- If-Modified-Since 比较资源更新时间（Last-Modified） ✨
- If-Unmodified-Since比较资源更新时间（Last-Modified），与 If-Modified-Since相反 ✨
- If-Rnages 资源未更新时发送实体byte的范围请求
- Range 实体的字节范围请求 ✨
- Authorization web的认证信息 ✨
- Proxy-Authorization 代理服务器要求web认证信息
- Host 请求资源所在服务器 ✨
- From 用户的邮箱地址
- User-Agent 客户端程序信息 ✨
- Max-Forwrads 最大的逐跳次数
- TE 传输编码的优先级

- Referer 请求原始放的url
- Expect 期待服务器的特定行为

响应首部字段（Response Header Fields）:从服务器向客户端响应时使用的字段

- Accept-Ranges 能接受的字节范围
- Age 推算资源创建经过时间
- Location 令客户端重定向的URI ✨
- vary 代理服务器的缓存信息
- ETag 能够表示资源唯一资源的字符串 ✨
- WWW-Authenticate 服务器要求客户端的验证信息
- Proxy-Authenticate 代理服务器要求客户端的验证信息
- Server 服务器的信息 ✨
- Retry-After 和状态码503 一起使用的首部字段，表示下次请求服务器的时间

实体首部字段（Entiy Header Fields）:针对请求报文和响应报文的实体部分使用首部

- Allow 资源可支持http请求的方法 ✨
- Content-Language 实体的资源语言
- Content-Encoding 实体的编码格式
- Content-Length 实体的大小（字节）
- Content-Type 实体媒体类型
- Content-MD5 实体报文的摘要
- Content-Location 代替资源的yri
- Content-Rnages 实体主体的位置返回
- Last-Modified 资源最后的修改资源 ✨
- Expires 实体主体的过期资源 ✨

聊一聊HTTP的状态码有哪些?

> 内容很多，重点看标『✨』内容

2XX 成功

- 200 OK，表示从客户端发来的请求在服务器端被正确处理 ✨
- 201 Created 请求已经被实现，而且有一个新的资源已经依据请求的需要而建立
- 202 Accepted 请求已接受，但是还没执行，不保证完成请求
- 204 No content，表示请求成功，但响应报文不含实体的主体部分
- 206 Partial Content，进行范围请求 ✨

3XX 重定向

- 301 moved permanently，永久性重定向，表示资源已被分配了新的 URL
- 302 found，临时性重定向，表示资源临时被分配了新的 URL ✨
- 303 see other，表示资源存在着另一个 URL，应使用 GET 方法丁香获取资源
- 304 not modified，表示服务器允许访问资源，但因发生请求未满足条件的情况
- 307 temporary redirect，临时重定向，和302含义相同

4XX 客户端错误

- 400 bad request，请求报文存在语法错误 ✨
- 401 unauthorized，表示发送的请求需要有通过 HTTP 认证的认证信息 ✨
- 403 forbidden，表示对请求资源的访问被服务器拒绝 ✨
- 404 not found，表示在服务器上没有找到请求的资源 ✨
- 408 Request timeout, 客户端请求超时
- 409 Confict, 请求的资源可能引起冲突

5XX 服务器错误

- 500 internal sever error，表示服务器端在执行请求时发生了错误 ✨
- 501 Not Implemented 请求超出服务器能力范围，例如服务器不支持当前请求所需要的某个功能，或者请求是服务器不支持的某个方法
- 503 service unavailable，表明服务器暂时处于超负载或正在停机维护，无法处理请求
- 505 http version not supported 服务器不支持，或者拒绝支持在请求中使用的 HTTP 版本

同样是重定向307，303，302的区别?

302是http1.0的协议状态码，在http1.1版本的时候为了细化302状态码又出来了两个303和307。

303明确表示客户端应当采用get方法获取资源，他会把POST请求变为GET请求进行重定向。 307会遵照浏览器标准，不会从post变为get。

HTTP的keep-alive是干什么的?

在早期的HTTP/1.0中，每次http请求都要创建一个连接，而创建连接的过程需要消耗资源和时间，为了减少资源消耗，缩短响应时间，就需要重用连接。在后来的HTTP/1.0中以及HTTP/1.1中，引入了重用连接的机制，就是在http请求头中加入Connection: keep-alive来告诉对方这个请求响应完成后不要关闭，下一次咱们还用这个请求继续交流。协议规定HTTP/1.0如果想要保持长连接，需要在请求头中加上Connection: keep-alive。

keep-alive的优点：

- 较少的CPU和内存的使用（由于同时打开的连接的减少了）
- 允许请求和应答的HTTP管线化
- 降低拥塞控制 （TCP连接减少了）
- 减少了后续请求的延迟（无需再进行握手）
- 报告错误无需关闭TCP连

HTTP2相对于HTTP1.x有什么优势和特点?

二进制分帧

帧：HTTP/2 数据通信的最小单位消息：指 HTTP/2 中逻辑上的 HTTP 消息。例如请求和响应等，消息由一个或多个帧组成。

流：存在于连接中的一个虚拟通道。流可以承载双向消息，每个流都有一个唯一的整数ID

HTTP/2 采用二进制格式传输数据，而非 HTTP 1.x 的文本格式，二进制协议解析起来更高效。

服务器推送

服务端可以在发送页面HTML时主动推送其它资源，而不用等到浏览器解析到相应位置，发起请求再响应。例如服务端可以主动把JS和CSS文件推送给客户端，而不需要客户端解析HTML时再发送这些请求。

服务端可以主动推送，客户端也有权利选择是否接收。如果服务端推送的资源已经被浏览器缓存过，浏览器可以通过发送RST_STREAM帧来拒收。主动推送也遵守同源策略，服务器不会随便推送第三方资源给客户端。

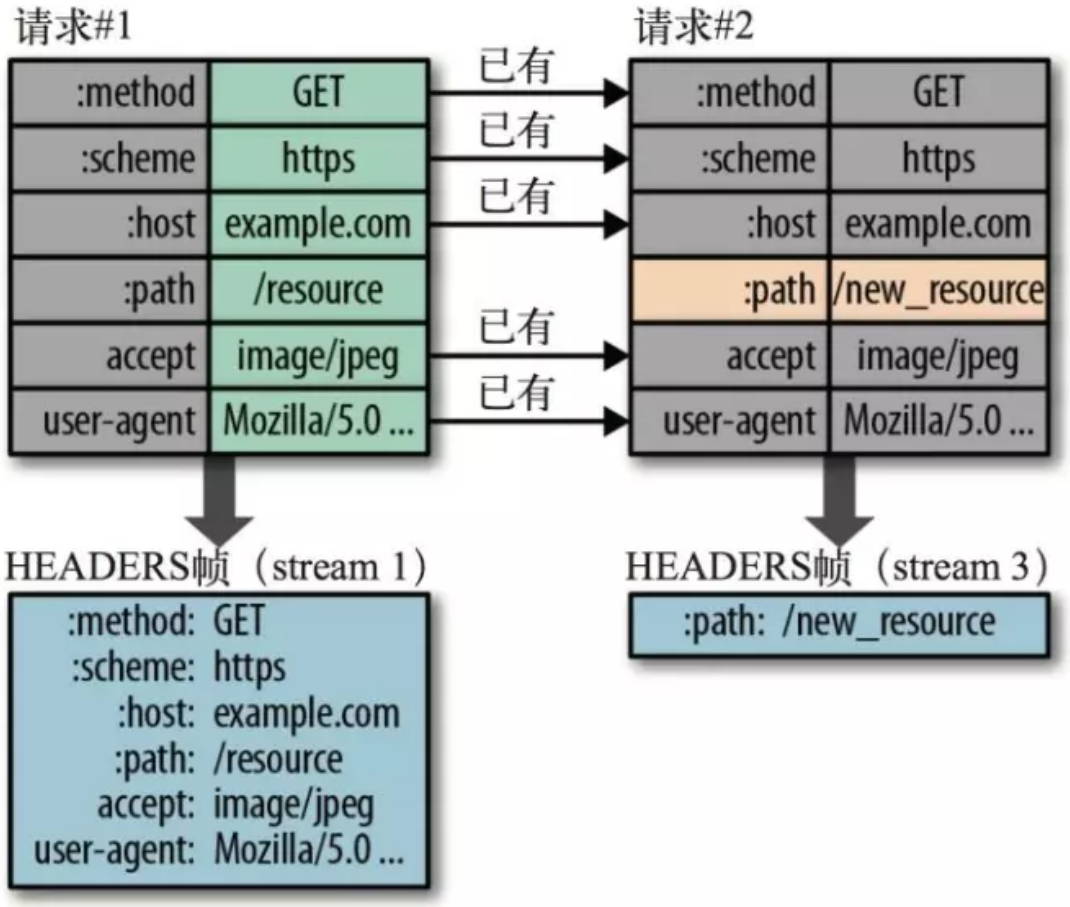
头部压缩

HTTP/1.x会在请求和响应中中重复地携带不常改变的、冗长的头部数据，给网络带来额外的负担。

- HTTP/2在客户端和服务器端使用"首部表"来跟踪和存储之前发送的键 - 值对，对于相同的数据，不再通过每次请求和响应发送

- 首部表在HTTP/2的连接存续期内始终存在，由客户端和服务器共同渐进地更新;
- 每个新的首部键 - 值对要么被追加到当前表的末尾，要么替换表中之前的值。

> 你可以理解为只发送差异数据，而不是全部发送，从而减少头部的信息量



多路复用

HTTP 1.x 中，如果想并发多个请求，必须使用多个 TCP 链接，且浏览器为了控制资源，还会对单个域名有 6-8个的TCP链接请求限制。

HTTP2中：

- 同域名下所有通信都在单个连接上完成。
- 单个连接可以承载任意数量的双向数据流。
- 数据流以消息的形式发送，而消息又由一个或多个帧组成，多个帧之间可以乱序发送，因为根据帧首部的流标识可以重新组装

HTTPS 是什么?

HTTPS ，实际就是在 TCP 层与 HTTP 层之间加入了 SSL/TLS 来为上层的安全保驾护航，主要用到对称加密、非对称加密、证书，等技术进行客户端与服务器的数据加密传输，最终达到保证整个通信的安全性。

为什么有了HTTP为什么还要HTTPS?

https是安全版的http，因为http协议的数据都是明文进行传输的，所以对于一些敏感信息的传输就很不安全，HTTPS就是为了解决HTTP的不安全而生的。

HTTPS是如何保证安全的?

过程比较复杂，我们得先理解两个概念

对称加密：即通信的双方都使用同一个秘钥进行加解密，比如特务接头的暗号，就属于对称加密

对称加密虽然很简单性能也好，但是无法解决首次把秘钥发给对方的问题，很容易被hacker拦截秘钥。

非对称加密：

1. 私钥 + 公钥= 密钥对
2. 即用私钥加密的数据,只有对应的公钥才能解密,用公钥加密的数据,只有对应的私钥才能解密
3. 因为通信双方的手里都有一套自己的密钥对,通信之前双方会先把自己的公钥都先发给对方
4. 然后对方再拿着这个公钥来加密数据响应给对方,等到到了对方那里,对方再用自己的私钥进行解密

非对称加密虽然安全性更高，但是带来的问题就是速度很慢，影响性能。

解决方案：

那么结合两种加密方式，将对称加密的密钥使用非对称加密的公钥进行加密，然后发送出去，接收方使用私钥进行解密得到对称加密的密钥，然后双方可以使用对称加密来进行沟通。

此时又带来一个问题，中间人问题：

如果此时在客户端和服务器之间存在一个中间人,这个中间人只需要把原本双方通信互发的公钥,换成自己的公钥,这样中间人就可以轻松解密通信双方所发送的所有数据。

所以这个时候需要一个安全的第三方颁发证书（CA），证明身份的身份，防止被中间人攻击。

证书中包括：签发者、证书用途、使用者公钥、使用者私钥、使用者的HASH算法、证书到期时间等

但是问题来了，如果中间人篡改了证书，那么身份证明是不是就无效了？这个证明就白买了，这个时候需要一个新的技术，数字签名。

数字签名就是用CA自带的HASH算法对证书的内容进行HASH得到一个摘要，再用CA的私钥加密，最终组成数字签名。

当别人把他的证书发过来的时候,我再用同样的Hash算法,再次生成消息摘要，然后用CA的公钥对数字签名解密,得到CA创建的消息摘要,两者一比,就知道中间有没有被人篡改了。

这个时候就能最大程度保证通信的安全了。

IO模型

nio和 bio 、aio的区别

消息时的系统通信，通常基于网络协议实现。常见的协议包括TCP/IP,UDP/IP。

TCP/IP等协议用于数据传输，但要完成通信，还需要对数据进行处理。例如读取和写入数据。

I/O可以分为两种：同步IO和异步IO,同步I/O最常见的是 BIO(Blocking IO)、NIO(Non-Blocking IO)

BIO：是当发起I/O的读或写操作时，均为阻塞方式，直到应用程序读到了流或者将流写入数据。

NIO：基于事件驱动思想，常采用reactor（反应器）模式。当发起 IO请求时，应用程序是非阻塞的。当SOCKET有流可读或写的时候，

由操作系统通知应用程序，应用程序再将流读取到缓冲区或者写入系统。

AIO：同样基于事件驱动的思想，通常采用Proactor(前摄器模式)实现。对于读操作，操作系统将数据读到缓冲区，并通知应用程序，对于写操作，操作系统将write方法传递的流写入并主动通知

应用程序。它节省了NIO中遍历事件通知队列的代价。

阻塞　某个请求发出后，由于该请求操作需要的条件不满足，请求操作一直阻塞，不会返回，直到条件满足。

非阻塞　请求发出后，若该请求需要的条件不满足，则立即返回一个标志信息告知条件不满足，而不会一直等待。一般需要通过循环判断请求条件是否满足来获取请求结果。

这里注意比较NIO和AIO的不同，AIO是操作系统完成IO并通知应用程序，NIO是操作系统通知应用程序，由应用程序完成。

reactor 模型

当客户端请求抵达后，服务处理程序使用多路分配策略，由一个非阻塞的线程来接收所有的请求，然后派发这些请求至相关的工作线程进行处理

TCP协议

网络的七层结构及其作用

自上而下是:

应用层（数据）：确定进程之间通信的性质以满足用户需要以及提供网络与用户应用

表示层（数据）：主要解决用户信息的语法表示问题，如加密解密

会话层（数据）：提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制，如服务器验证用户登录便是由会话层完成的

传输层（段）：实现网络不同主机上用户进程之间的数据通信，可靠

与不可靠的传输，传输层的错误检测，流量控制等

网络层（包）：提供逻辑地址（IP）、选路，数据从源端到目的端的

传输

数据链路层（帧）：将上层数据封装成帧，用MAC地址访问媒介，错误检测与修正

物理层（比特流）：设备之间比特流的传输，物理接口，电气特性等

TCP协议

TCP/IP协议按照层次分为以下四层。应用层、传输层、网络层、数据链路层。

TCP（Transmission Control Protocol，传输控制协议）是面向连接的协议，也就是说，在收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来，其中的过程非常复杂，

TCP选项有哪些

TCP首部选项字段多达40B，一些常用的字段有：

1）选项结束字段（EOP，0x00），占1B，一个报文段仅用一次。放在末尾用于填充，用途是说明：首部已经没有更多的消息，应用数据在下一个32位字开始处

2）无操作字段（NOP, 0x01），占1B，也用于填充，放在选项的开头

3）MSS（最大报文段长度），格式如下：种类（1B，值为2），长度（1B，值为4），数值（2B）

用于在连接开始时确定MSS的大小，如果没有确定，就用默认的（一般实现是536B）

4）窗口扩大因子，格式如下：种类（1B，值为3），长度（1B，值为3),数值（1B）

新窗口值 = 首部窗口值 * 2的（扩大因子）次方

当通信双方认为首部的窗口值还不够大的时候，在连接开始时用这个来定义更大的窗口。仅在连接开始时有效。一经定义，通信过程中无法更改。

5）时间戳（应用测试RTT和防止序号绕回）

6）允许SACK和SACK选项

2.4.1 出现RST包的原因

RST：TCP首部中的6个标志比特之一,表示重置连接、复位连接。

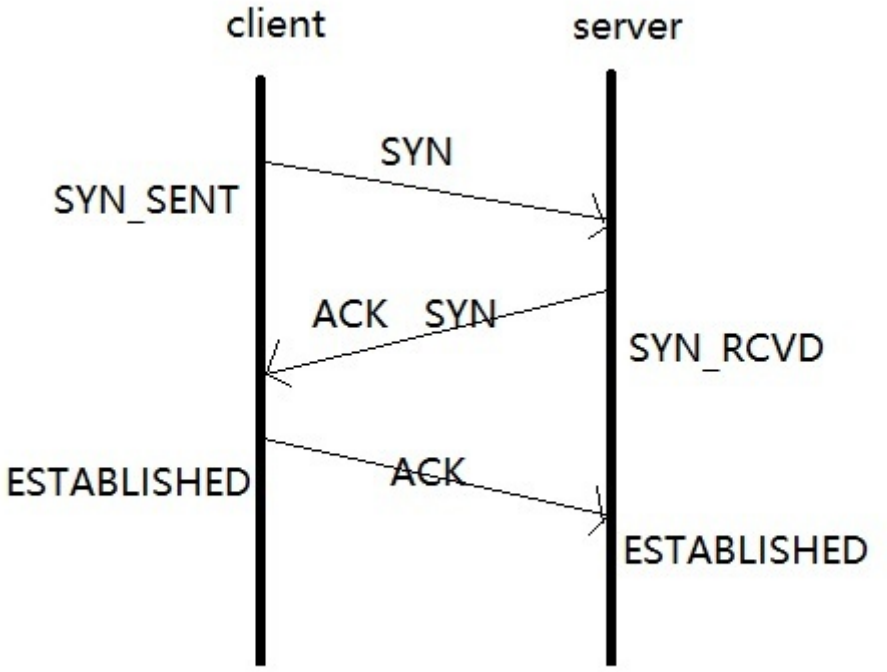
1. 服务器端口未打开而客户端来连接时
2. 在一个已关闭的SOCKET上收到数据
3. 请求超时
4. 提前关闭

TCP三次握手：

发送方：我要和你建立链接?

接收方：你真的要和我建立链接么?

发送方：我真的要和你建立链接，成功。



为什么 TCP 连接需要三次握手，两次不可以么，为什么?

为了防止已失效的连接请求报文突然又传送到了服务端，因而产生错误。客户端发出的连接请求报文并未丢失，而是在某个网络节点长时间滞留了，以致延误到链接释放以后的某个时间才到达 Server 。

问:三次握手过程中有哪些不安全性

一、SYN洪泛攻击

服务器处于SYN_Wait的状态：

洪泛攻击

那么在这段时间内，可能会有SYN Flood的风险（洪泛攻击）。黑客就会利用伪造的IP地址像服务端不断的发送请求，却不发送ACk，那么本次的TCP连接就处于挂起的状态，服务端在等待关闭该连接的过程中耗尽了资源（收不到客户端的ACK，就会重复发送Syn-Ack，消耗内存和CPU，直至服务器宕机），正常的客户端发送的请求就得不到回应。

SYN Flood的防护措施

当SYN队列满了之后，通过tcp_syncookies参数回发SYN Cookie，（tcp_syncookies是通过源IP地址端口，目的地址端口和时间戳来确定的一个sequnce number）只有正常的客户端收到该cookie之后会回发 Syn Cookie，直接和服务端建立连接。

通过Syn cookie，即使当前的队列满了，当前连接不在队列中，依然可以建立连接。

1）伪装的IP向服务器发送一个SYN请求建立连接，然后服务器向该IP回复SYN和ACK，但是找不到该IP对应的主机，当超时时服务器收不到ACK会重复发送。当大量的攻击者请求建立连接时，服务器就会存在大量未完成三次握手的连接，服务器主机backlog被耗尽而不能响应其它连接。**即SYN泛洪攻击**（属于DOS的一种，发送大量的半连接请求，耗费CPU和内存资源，引起网络堵塞甚至系统瘫痪）

当你在服务器上看到大量的半连接状态时，特别是源IP地址是随机的，基本上可以断定这是一次SYN攻击.在Linux下可以如下命令检测是否被Syn攻击

```
netstat -n -p TCP | grep SYN_RECV
```

防范措施：

1、**降低SYN timeout时间**，使得主机尽快释放半连接的占用

2、**采用SYN cookie设置**，如果短时间内连续收到某个IP的重复SYN请求，则认为受到了该IP的攻击，丢弃来自该IP的后续请求报文

3、在网关处设置**过滤**，拒绝将一个源IP地址不属于其来源子网的包进行更远的路由

2）当一个主机向服务器发送SYN请求连接，服务器回复ACK和SYN后，攻击者截获ACK和SYN。然后伪装成原始主机继续与服务器进行通信。

二、DOS攻击 拒绝服务攻击

DDoS（分布式拒绝服务攻击）

DOS攻击利用合理的服务请求占用过多的服务资源，使正常用户的请求无法得到相应。

常见的DOS攻击有计算机网络带宽攻击和连通性攻击。

带宽攻击指以极大的通信量冲击网络，使得所有可用网络资源都被消耗殆尽，最后导致合法的用户请求无法通过。

连通性攻击指用大量的连接请求冲击计算机，使得所有可用的操作系统资源都被消耗殆尽，最终计算机无法再处理合法用户的请求。

三、死亡值ping

许多操作系统的TCP/IP协议栈规定**ICMP包大小为64KB**（网间控制报文），且在对包的标题头进行读取之后，要根据该标题头里包含的信息来为有效载荷生成缓冲区。

”死亡值ping”就是**故意产生畸形的测试ping包**，声称自己的尺寸超过ICMP上限，也就是加载的尺寸超过64KB上限，使未采取保护措施的网络系统出现内存分配错误，**导致TCP/IP协议栈崩溃，最终接收方宕机。**

PING (Packet Internet Groper)，因特网包探索器，用于测试网络连通性的程序。

Ping发送一个ICMP(Internet Control Messages Protocol,因特网信报控制协议)；回声请求消息给目的地并报告是否收到所希望的ICMPecho （ICMP回声应答）。

工作原理：利用网络上机器IP地址的唯一性，给目标IP地址发送一个数据包，再要求对方返回一个同样大小的数据包来确定两台网络机器是否连接相通，时延是多少。

为了提高IP数据报交付成功机会，在网络层使用过了网际控制报文协议（ICMP）来允许主机或路由器报告差错和异常情况。ICMP报文作为IP层数据报的数据，加上数据报首部，组成IP数据报发出去。ICMP协议是IP层协议。

保活机制

保活机制 由一个保活计数器实现。是服务器用来确认什么时候应该断开连接的一种机制。
保活功能可以被设置在TCP连接的一端、两端,或者两端都没有。

如果建立了连接之后，客户端出现了故障怎么办?
保活机制：在keep alive time保活时间内处于非活动状态，那么开启保活机制的一端向对方发送保活探测报文，如果（在保活时间间隔内keep alive interval）没有收到响应就继续发送，直到达到了保活探测的次数(keep alive probe)，对方被确认为不可达，本次连接中断。

什么是 TCP 四次挥手?

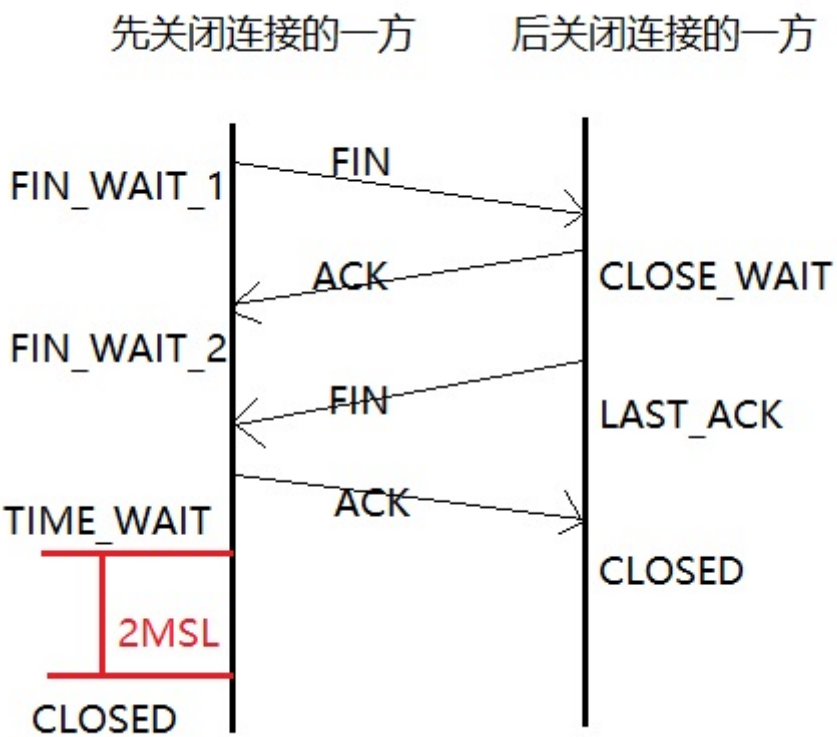
四次挥手，简单来说，就是：

发送方：我要和你断开连接！

接收方：好的，断吧。

接收方：我也要和你断开连接！

发送方：好的，断吧。



问题：为什么要有TIME_WAIT状态?

TIME_WAIT状态存在有两个原因。
一、可靠终止TCP连接。如果最后一个ACK报文因为网络原因被丢弃，此时server因为没有收到ACK而超时重传FIN报文，处于TIME_WAIT状态的client可以继续对FIN报文做回复，向server发送ACK报文。

二、保证让迟来的TCP报文段有足够的时间被识别和丢弃。连接结束了，网络中的延迟报文也应该被丢弃掉，以免影响立刻建立的新连接。

问题：TCP粘包、拆包及解决办法

为什么常说 TCP 有粘包和拆包的问题而不说 UDP ?

由前两节可知，UDP 是基于报文发送的，UDP首部采用了 16bit 来指示 UDP 数据报文的长度，因此在应用层能很好的将不同的数据报文区分开，从而避免粘包和拆包的问题。

而 TCP 是基于字节流的，虽然应用层和 TCP 传输层之间的数据交互是大小不等的数据块，但是 TCP 并没有把这些数据块区分边界，仅仅是一连串没有结构的字节流；另外从 TCP 的帧结构也可以看出，在 TCP 的首部没有表示数据长度的字段，基于上面两点，在使用 TCP 传输数据时，才有粘包或者拆包现象发生的可能。

什么是粘包、拆包?

假设 Client 向 Server 连续发送了两个数据包，用 packet1 和 packet2 来表示，那么服务端收到的数据可以分为三种情况，现列举如下：

第一种情况，接收端正常收到两个数据包，即没有发生拆包和粘包的现象。

第二种情况，接收端只收到一个数据包，但是这一个数据包中包含了发送端发送的两个数据包的信息，这种现象即为粘包。这种情况由于接收端不知道这两个数据包的界限，所以对于接收端来说很难处理。

第三种情况，这种情况有两种表现形式，如下图。接收端收到了两个数据包，但是这两个数据包要么是不完整的，要么就是多出来一块，这种情况即发生了拆包和粘包。这两种情况如果不加特殊处理，对于接收端同样是不好处理的。

为什么会发生 TCP 粘包、拆包?

- 要发送的数据大于 TCP 发送缓冲区剩余空间大小，将会发生拆包。
- 待发送数据大于 MSS（最大报文长度），TCP 在传输前将进行拆包。
- 要发送的数据小于 TCP 发送缓冲区的大小，TCP 将多次写入缓冲区的数据一次发送出去，将会发生粘包。
- 接收数据端的应用层没有及时读取接收缓冲区中的数据，将发生粘包。

粘包、拆包解决办法

由于 TCP 本身是面向字节流的，无法理解上层的业务数据，所以在底层是无法保证数据包不被拆分和重组的，这个问题只能通过上层的应用协议栈设计来解决，根据业界的主流协议的解决方案，归纳如下：

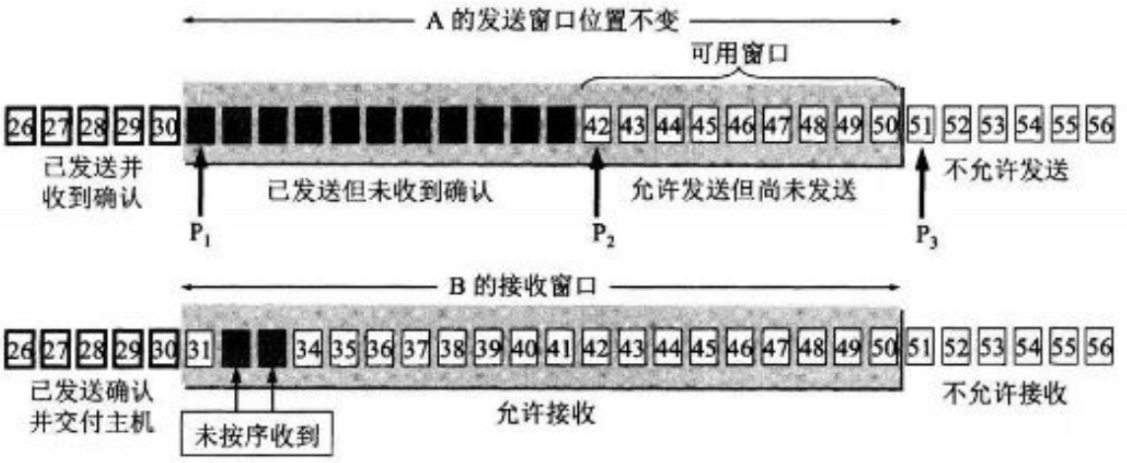
- **消息定长**：发送端将每个数据包封装为固定长度（不够的可以通过补 0 填充），这样接收端每次接收缓冲区中读取固定长度的数据就自然而然的把每个数据包拆分开来。
- **设置消息边界**：服务端从网络流中按消息边界分离出消息内容。在包尾增加回车换行符进行分割，例如 FTP 协议。
- **将消息分为消息头和消息体**：消息头中包含表示消息总长度（或者消息体长度）的字段。
- 更复杂的应用层协议比如 Netty 中实现的一些协议都对粘包、拆包做了很好的处理。

问题：说说TCP 滑动窗口

窗口是缓存的一部分，用来暂时存放字节流。发送方和接收方各有一个窗口，接收方通过 TCP 报文段中的窗口字段告诉发送方自己的窗口大小，发送方根据这个值和其它信息设置自己的窗口大小。

发送窗口内的字节都允许被发送，接收窗口内的字节都允许被接收。如果发送窗口左部的字节已经发送并且收到了确认，那么就将发送窗口向右滑动一定距离，直到左部第一个字节不是已发送并且已确认的状态；接收窗口的滑动类似，接收窗口左部字节已经发送确认并交付主机，就向右滑动接收窗口。

接收窗口只会对窗口内最后一个按序到达的字节进行确认，例如接收窗口已经收到的字节为 {31, 34, 35}，其中 {31} 按序到达，而 {34, 35} 就不是，因此只对字节 31 进行确认。发送方得到一个字节的确认之后，就知道这个字节之前的所有字节都已经被接收。



说说TCP 流量控制?

流量控制是为了控制发送方发送速率，保证接收方来得及接收。

接收方发送的确认报文中的窗口字段可以用来控制发送方窗口大小，从而影响发送方的发送速率。将窗口字段设置为 0，则发送方不能发送数据。

实际上，为了避免此问题的产生，发送端主机会时不时的发送一个叫做窗口探测的数据段，此数据段仅包含一个字节来获取最新的窗口大小信息。

UDP 协议

UDP 是什么?

UDP（User Data Protocol，用户数据报协议），是与 TCP 相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去。

问题：UDP 主要特点

UDP 是无连接的。

UDP 使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的链接状态（这里面有许多参数）。

UDP 是面向报文的。

UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低。

tcp与UDP的区别

TCP 是面向连接的；UDP 是无连接的。

TCP 是可靠的；UDP 是不可靠的。

TCP 只支持点对点通信；UDP 支持一对一、一对多、多对一、多对多的通信模式。

TCP 是面向字节流的；UDP 是面向报文的。

TCP 有拥塞控制机制；UDP 没有拥塞控制，适合媒体通信。

TCP 首部开销(20 个字节)，比 UDP 的首部开销(8 个字节)要大。

问题：UDP 和 TCP 的特点与区别

用户数据报协议 UDP（User Datagram Protocol）

是无连接的，尽最大可能交付，没有拥塞控制，面向报文（对于应用程序传下来的报文不合并也不拆分，只是添加 UDP 首部），支持一对一、一对多、多对一和多对多的交互通信。

传输控制协议 TCP（Transmission Control Protocol）

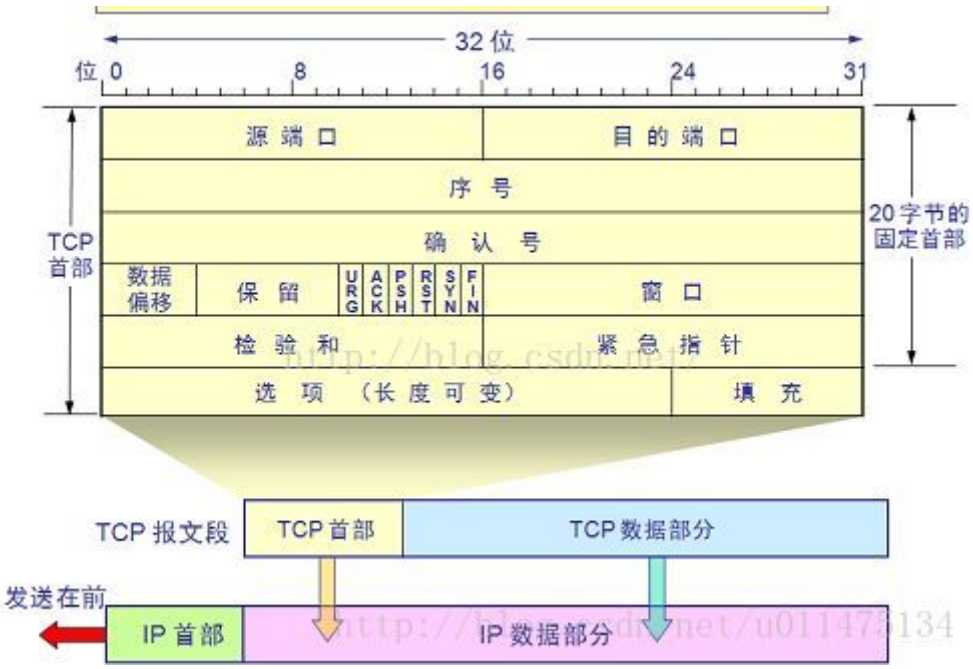
是面向连接的，提供可靠交付，有流量控制，拥塞控制，提供全双工通信，面向字节流（把应用层传下来的报文看成字节流，把字节流组织成大小不等的数据块），每一条 TCP 连接只能是点对点的（一对一）。

问题：对比一下UDP 、TCP 首部格式?

UDP 首部字段如下：

UDP 首部字段只有 8 个字节，包括源端口、目的端口、长度、检验和。12 字节的伪首部是为了计算检验和临时添加的。

TCP 首部字段如下：



TCP 首部格式比 UDP 复杂。

序号：用于对字节流进行编号，例如序号为 301，表示第一个字节的编号为 301，如果携带的数据长度为 100 字节，那么下一个报文段的序号应为 401。

确认号：期望收到的下一个报文段的序号。例如 B 正确收到 A 发送来的一个报文段，序号为 501，携带的数据长度为 200 字节，因此 B 期望下一个报文段的序号为 701，B 发送给 A 的确认报文段中确认号就为 701。

数据偏移：指的是数据部分距离报文段起始处的偏移量，实际上指的是首部的长度。

控制位：八位从左到右分别是 CWR，ECE，URG，ACK，PSH，RST，SYN，FIN。

CWR：CWR 标志与后面的 ECE 标志都用于 IP 首部的 ECN 字段，ECE 标志为 1 时，则通知对方已将拥塞窗口缩小；

ECE：若其值为 1 则会通知对方，从对方到这边的网络有阻塞。在收到数据包的 IP 首部中 ECN 为 1 时将 TCP 首部中的 ECE 设为 1；

URG：该位设为 1，表示包中有需要紧急处理的数据，对于需要紧急处理的数据，与后面的紧急指针有关；

ACK：该位设为 1，确认应答的字段有效，TCP规定除了最初建立连接时的 SYN 包之外该位必须设为 1；

PSH：该位设为 1，表示需要将收到的数据立刻传给上层应用协议，若设为 0，则先将数据进行缓存；

RST：该位设为 1，表示 TCP 连接出现异常必须强制断开连接；

SYN：用于建立连接，该位设为 1，表示希望建立连接，并在其序列号的字段进行序列号初值设定；

FIN：该位设为 1，表示今后不再有数据发送，希望断开连接。当通信结束希望断开连接时，通信双方的主机之间就可以相互交换 FIN 位置为 1 的 TCP 段。

每个主机又对对方的 FIN 包进行确认应答之后可以断开连接。不过，主机收到 FIN 设置为 1 的 TCP 段之后不必马上回复一个 FIN 包，而是可以等到缓冲区中的所有数据都因为已成功发送而被自动删除之后再发 FIN 包；

窗口：窗口值作为接收方让发送方设置其发送窗口的依据。之所以要有这个限制，是因为接收方的数据缓存空间是有限的。

字节一面：在浏览器地址栏输入一个 URL 后回车，背后发生了什么?

说在前面

近段时间，有小伙伴面试字节，说遇到一个面试题：

> 在浏览器地址栏输入一个 URL 后回车，背后发生了什么?

这里尼恩给大家做一下系统化、体系化的梳理，使得大家可以充分展示一下大家雄厚的 “技术肌肉”，让面试官爱到 “不能自已、口水直流”。

也一并把这个 题目以及参考答案，收入咱们的《尼恩Java面试宝典》，供后面的小伙伴参考，提升大家的 3高 架构、设计、开发水平。

> 注：本文以 PDF 持续更新，最新尼恩 架构笔记、面试题 的PDF文件，请从这里获取：码云

0. 前言

在浏览器的地址栏输入一个 URL 后回车，背后到底发生了什么才能使得一个界面完美的展现在我们眼前?

今天讲解的这道题目，由于其涉及大量网络协议，可以非常直观的看出诸位小伙伴对计算机网络体系的整体把握程度，所以自然成为了各大公司的面试常客。

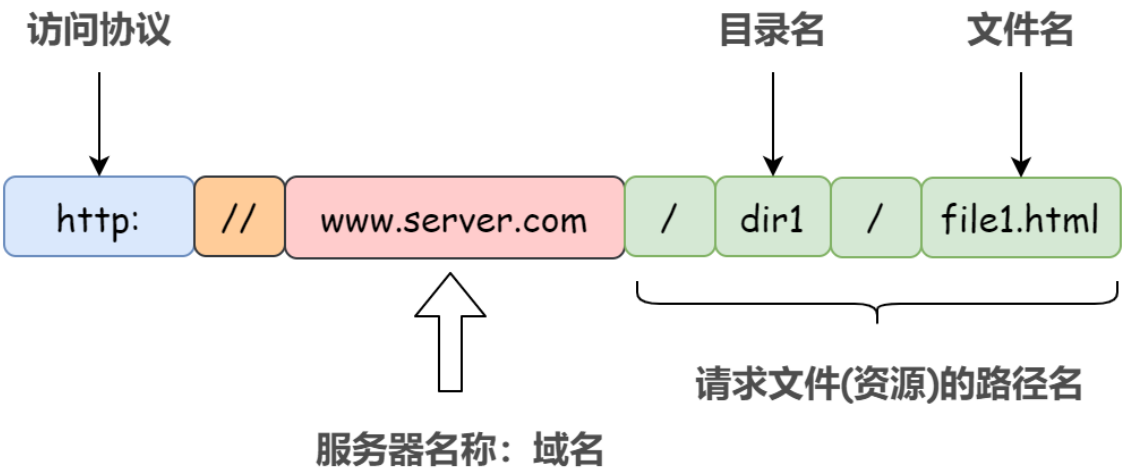
在浏览中输入 URL 并且获取响应的过程，其实就是浏览器和该 URL 对应的服务器的网络通信过程。比如我们输入 `www.baidu.com`，那么会返回一个百度搜索的界面，这其实就是浏览器和百度服务器之间的网络通信过程。浏览器就是客户端，用于发出请求，而百度的服务器就是服务端，用于接收并响应请求。

下面我们就来详细讲解这个庞大的网络通信过程。

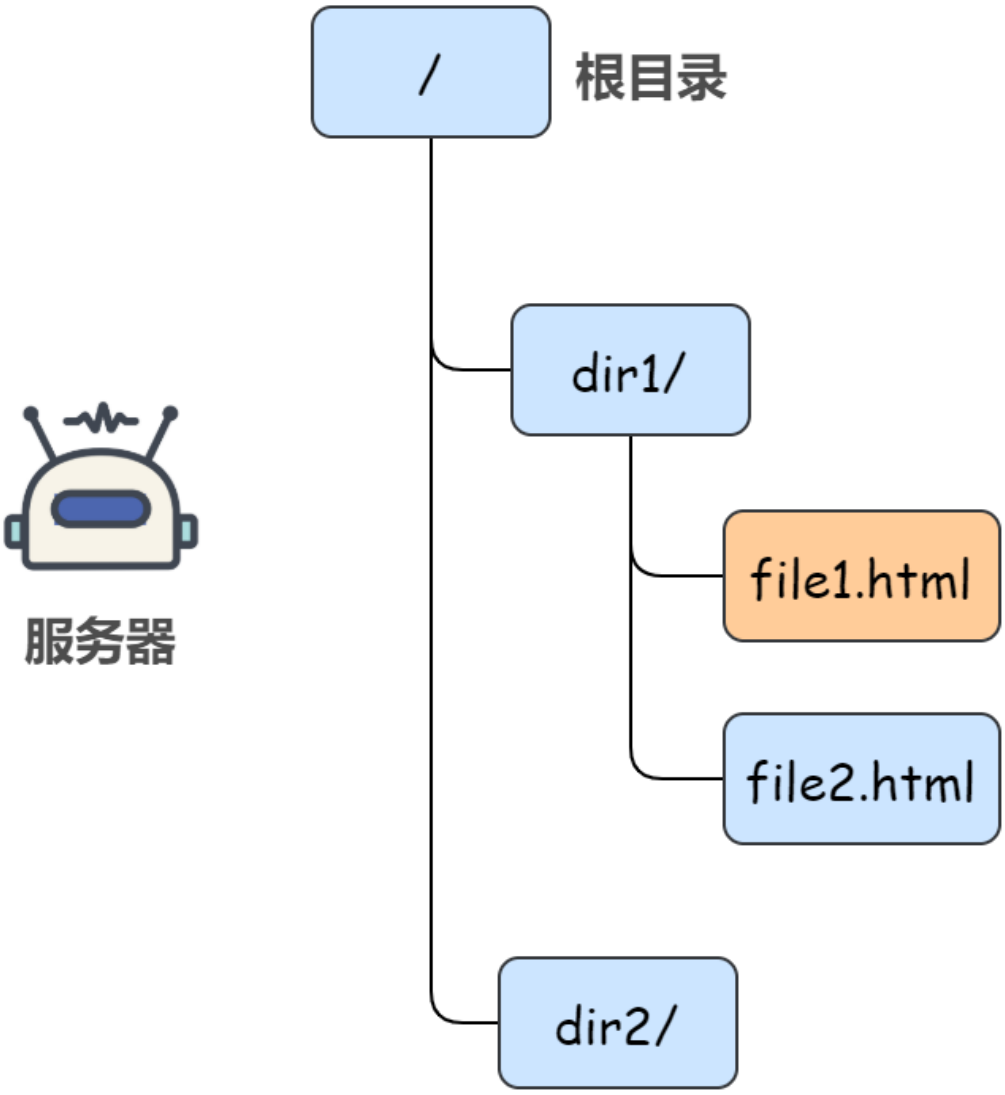
1. 解析 URL

不知道有没有同学会混淆域名和 URL 的概念，可以这样理解，URL 就是我们输入的网址，而网址里面含有域名。举个例子：`www.baidu.com/veal98` 是一个网址，而 `www.baidu.com` 就是服务器的域名。

URL 各元素的组成如下（当然，下述请求文件的路径名可以省略）：



这个 URL 请求的目标服务器上的文件路径就是：

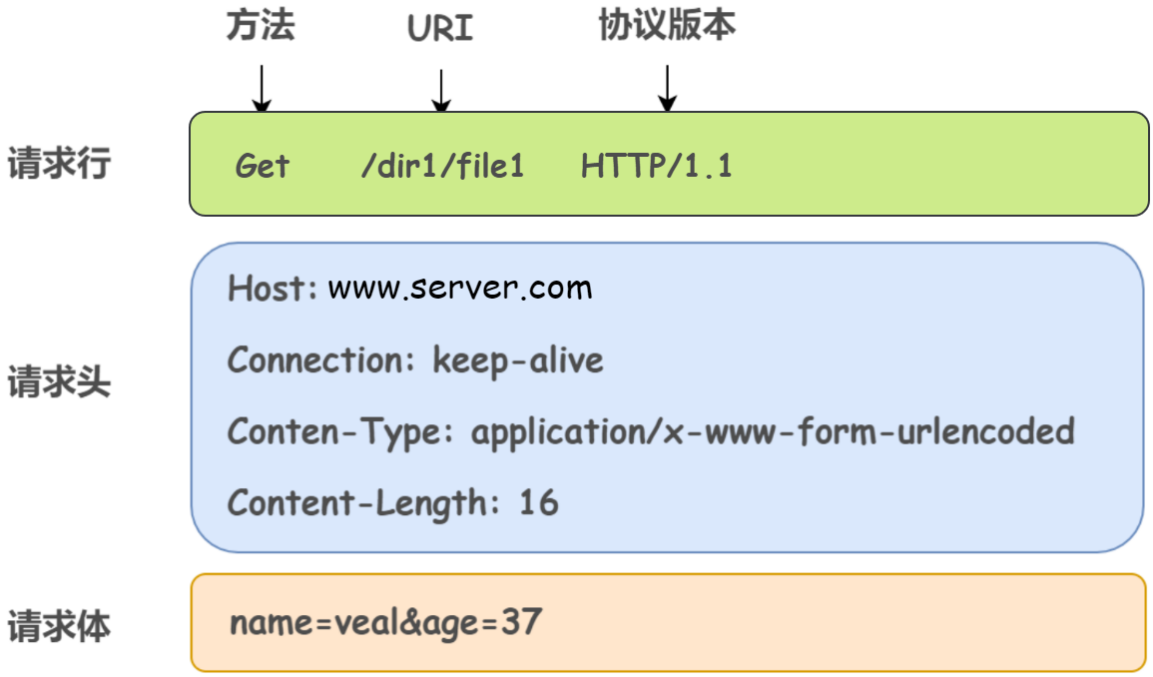


那么首先，浏览器做的第一步就是解析 URL 得到里面的参数，将域名和需要请求的资源分离开来，从而了解需要请求的是哪个服务器，请求的是服务器上什么资源等等。

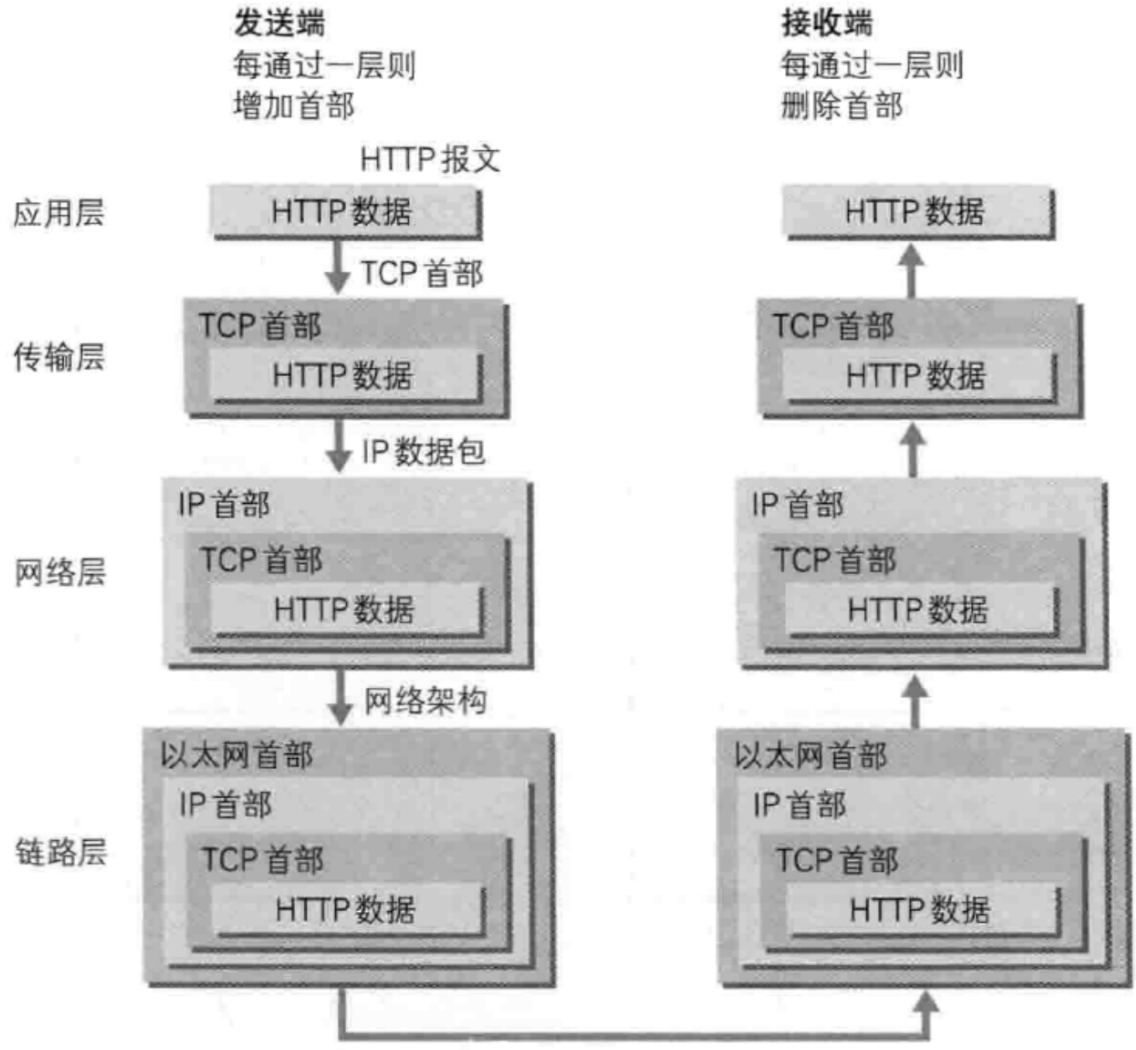
2. 浏览器封装 HTTP 请求报文

对 `URL` 进行解析之后，浏览器确定了目标服务器和文件名，接下来就需要根据这些消息**封装**成一个 HTTP 请求报文发送出去。举个 HTTP 请求报文的例子：

> 关于 HTTP 协议详细可见 《Java高并发核心编程（卷1）加强版》 这里不再赘述



解释一下**封装**，这是一个贯穿整个计算机网络的概念。就是说发送端在层与层之间传输数据时，每经过一层必定会被打上一个该层所属的首部信息。反之，接收端在层与层之间传输数据时，每经过一层就会把该层对应的首部信息消去。



3. DNS 域名解析获取 IP 地址

封装好 HTTP 请求报文后，在正式还有一项准备工作没有做，那就是获取目标服务器的 IP 地址。

虽然解析得到了域名，理论浏览器已经知道目标服务器是谁了。但是实际上，域名并不是目标服务器真正意义上的地址，互联网上每一台计算机都被全世界唯一 IP 地址标识着，但是 IP 地址并不方便记忆，所以才设计出了域名。

那么就需要解析域名获取目标服务器的 IP 地址。不然空有一个方便记忆的域名咋知道这个请求到底发送到哪里去呢。由域名转换得到 IP 地址就是 DNS 协议做的事情，如下：

> 关于 DNS 详细的内容各位可以回顾 DNS图解（秒懂 + 史上最全） 这篇文章，比如什么是域名，域名服务器，递归查询和迭代查询等等，写的已经足够详细，此处只列出 DNS 的解析过程。

1）首先搜索**浏览器的 DNS 缓存**，缓存中维护着一张域名与 IP 地址的对应表；

2）若没有命中，则继续搜索**操作系统的 DNS 缓存**；

3）若仍然没有命中，则操作系统将域名发送至**本地域名服务器**，本地域名服务器查询自己的 DNS 缓存，查找成功则返回结果（注意：主机和本地域名服务器之间的查询方式是**递归查询**）；

4）若本地域名服务器的 DNS 缓存没有命中，则本地域名服务器向上级域名服务器进行查询，通过以下方式进行**迭代查询**（注意：本地域名服务器和其他域名服务器之间的查询方式是迭代查询，防止根域名服务器压力过大）：

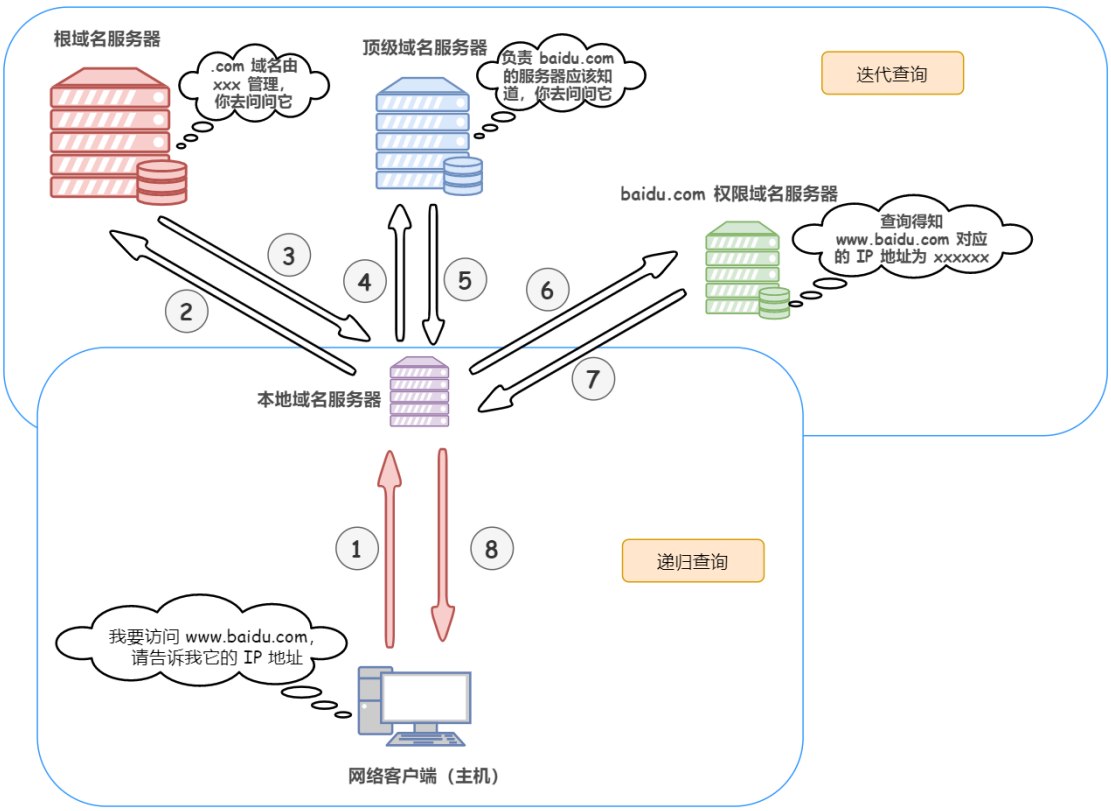
- 首先本地域名服务器向**根域名服务器**发起请求，根域名服务器是最高层次的，它并不会直接指明这个域名对应的 IP 地址，而是返回顶级域名服务器的地址，也就是说给本地域名服务器指明一条道路，让他去这里寻找答案
- 本地域名服务器拿到这个**顶级域名服务器**的地址后，就向其发起请求，获取**权限域名服务器**的地址
- 本地域名服务器根据权限域名服务器的地址向其发起请求，最终得到该域名对应的 IP 地址

4）本地域名服务器将得到的 IP 地址返回给操作系统，同时自己将 IP 地址缓存起来

5）操作系统将 IP 地址返回给浏览器，同时自己也将 IP 地址缓存起来

6）至此，浏览器就得到了域名对应的 IP 地址，并将 IP 地址缓存起来

配合下图直观理解：



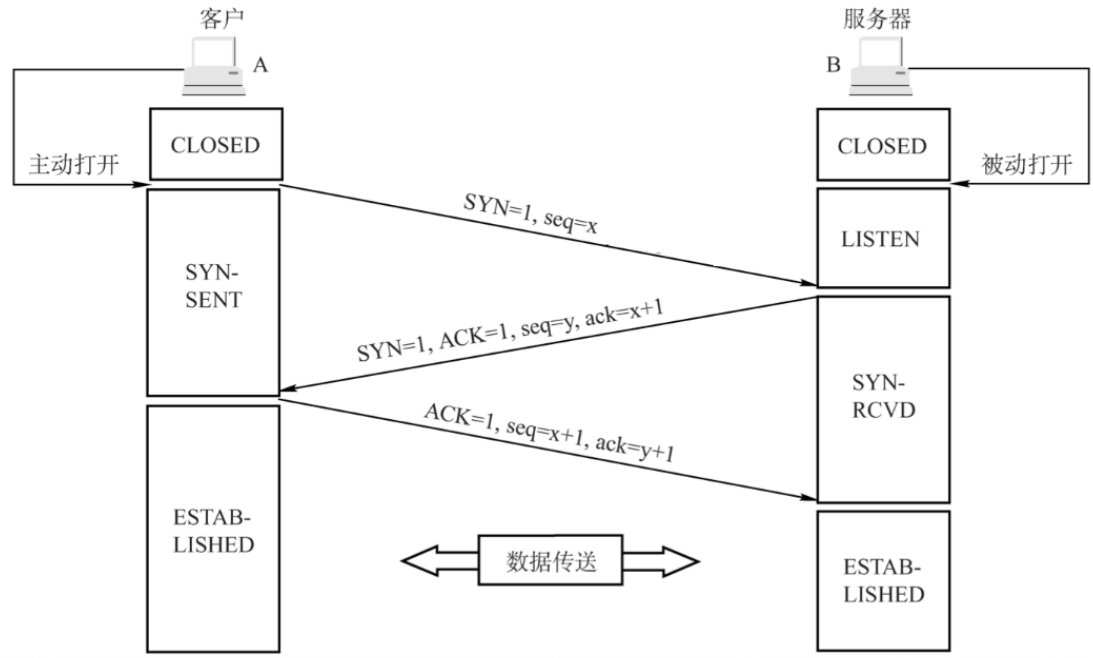
需要注意的是，DNS 使用的是 UDP 协议，也就是说上面各种请求的转发，都是基于 UDP 这个无连接协议的。

4. 建立 TCP 连接

获取到了目标服务器的 IP 地址之后，浏览器就知道我等下请求要发给谁了，这个时候就可以开始发送封装好了的 HTTP 请求报文了，那么既然需要发送请求，必然就需要 TCP 通过三次握手为浏览器和服务器之间建立可靠的连接，**保证双方都具有可靠的接收和发送能力。**

> 这里又是一道经典的面试题：TCP 三次握手和四次挥手，详细可见 关于 TCP 三次握手和四次挥手，满分回答在此 这篇文章。

三次握手过程如下图：



5. 浏览器发送请求

TCP 三次握手完成后，浏览器与目标服务器之间就建立了一个可靠的虚拟通道，于是浏览器就可以发送自己的 HTTP 请求了。

需要注意的是，HTTP 请求报文或者响应报文在 TCP 连接通道上进行传输的时候，由于这些报文比较大，为了更容易和准确可靠的传输，**TCP 会将 HTTP 报文按序号分割成若干报文段并加上 TCP 首部，分别进行传输。接收方在收到这些报文段后，按照序号以原来的顺序重组 HTTP 报文。**

6. 负责传输的 IP 协议

实际上，TCP 在三次握手建立连接、四次握手断开连接、以及连接建立过程中的收发数据（TCP 报文段）等各阶段操作时，都是通过 IP 协议进行传输的，IP 协议将这些阶段的数据添加 IP 首部封装成 IP 数据报再进行传输。

IP 数据报的首部存有**源 IP 地址**和 **目标 IP 地址。**所谓源 IP 地址 就是发送方的 IP 地址；目标 IP 地址就是通过 DNS 域名解析得到的目标服务器的 IP 地址。

事实上，**IP 协议身处的网络层规定的是：数据报要通过怎样的路径（传输路线）才能到达对方计算机，并传送给对方。**不理解这句话的详细解释马上就来，继续往下读。

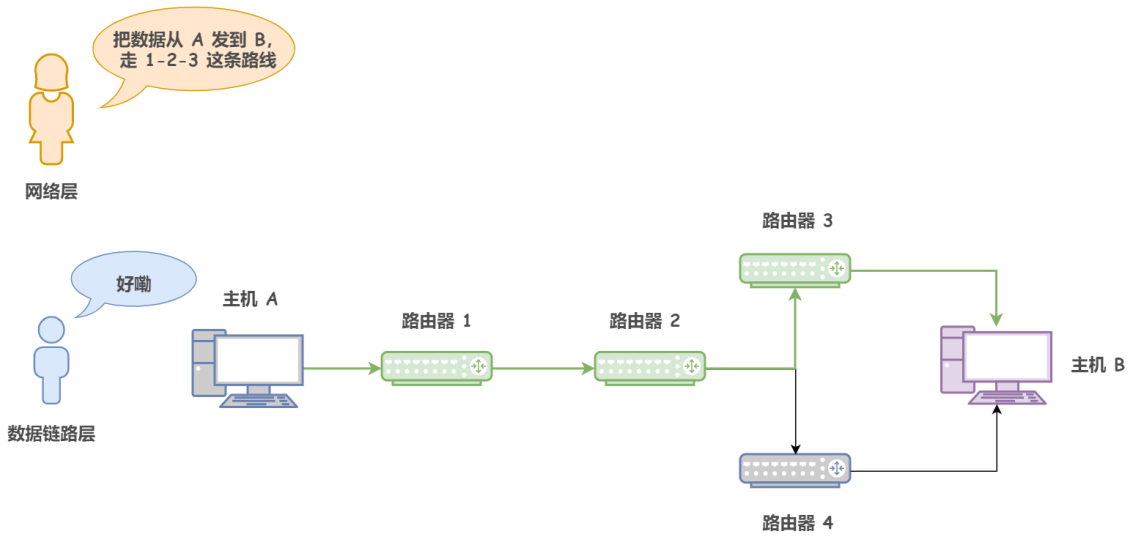
7. 使用 ARP 协议凭借 MAC 地址通信

> 关于 IP 协议、IP 地址、MAC 地址等详细请看 TCP/IP协议 （图解+秒懂+史上最全） 这篇文章。

上面说了，IP 协议的作用是把各种数据包传送给对方，而要保证确实传送到对方那里，则需要满足各类条件，其中必要的两个就是 IP 地址 和 MAC 地址。

MAC 地址也是用来唯一标识一个接入互联网的设备的，可能不禁有小伙伴要问，既然网络层已经有了唯一标识的 IP 地址，为啥还需要 MAC 地址？

看下面这幅图，在网络上，**通信的双方在同一局域网内的情况是很少见的，通常是需要多台计算机和网络设备的中转才能连接到对方。而在进行中转时，就需要利用下一站中转设备的 MAC 地址来搜索下一个中转目标。**

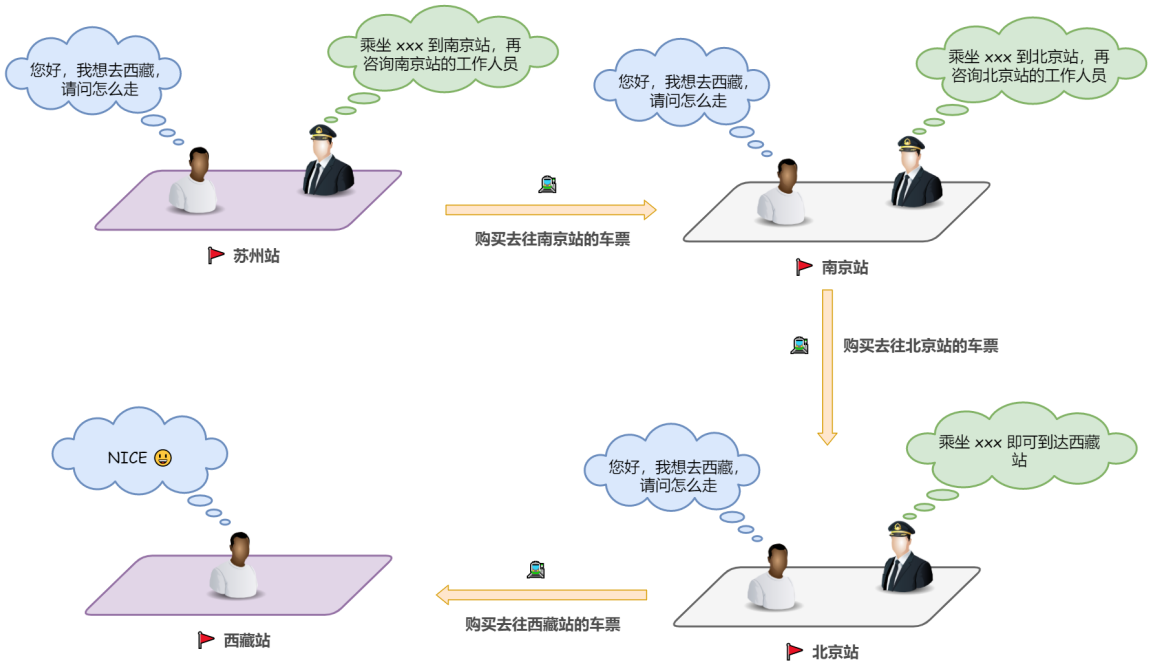


- 网络层指定了从哪个主机（「源 IP 地址」）发送到哪个主机（「目的 IP 地址」）。**源 IP 地址和目标 IP 地址在传输过程中是不会变化的**
- 而数据链路层则是根据 MAC 地址在一个接一个的区间中进行传输的，每个区间内的出发地址即「源 MAC 地址」，每个区间内的目的地址即「目的 MAC 地址」。显然，随着数据的传输，**源 MAC 地址和目的 MAC 地址会不断的发生变化**

  比如上图，**网络层告知了 1-2-3 路线，也就是说指明了这几个路由器的 IP 地址。那么数据链路层就会根据这几个 IP 地址对应的 MAC 地址依次找到 1、2、3，并在他们之间传输数据。**

这么说吧，举个形象点的例子：我们把数据链路层当成乘坐高铁从苏州到南京，再在南京转乘到北京，再在北京转乘到西藏的旅客，那么网络层就相当于每个车站的工作人员，**在数据链路层每次转乘时，网络层为其购买了一张标有下一个 MAC 地址的车票。**因此，即使旅客（数据链路层）不知道其最终目的地也没有关系，工作人员（网络层）会给你做出指引。

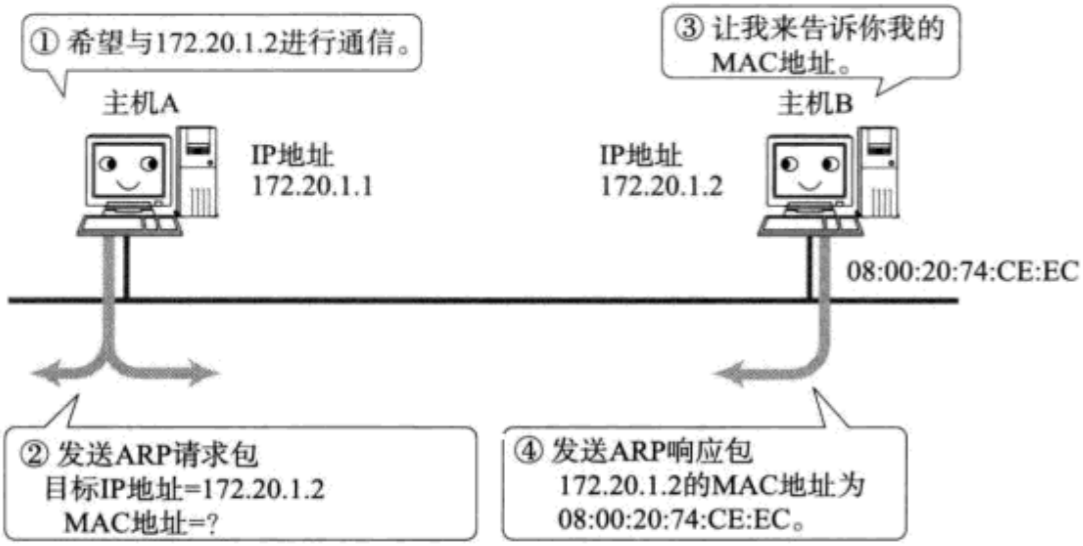
实际上，网络层做出指引的过程，我们将其称为**路由控制**，其中又涉及到了路由协议比如 OSPF 等



那么，**将 IP 地址转化为 MAC 地址**，从而在数据链路层精确的传输数据的协议就是 **ARP 协议。**

ARP 是借助 **ARP 请求与 ARP 响应**两种类型的包确定 MAC 地址的。并且每个主机都有一个 **ARP 高速缓存**，里面有本局域网上的各主机和路由器的 **IP 地址到 MAC 地址的映射表。**

如下图所示，假定主机 A 向同一链路上的主机 B 发送 IP 数据报，已知主机 A 和主机 B 的 IP 地址，它们互不知道对方的 MAC 地址：



1）首先，主机 A 为了获得主机 B 的 MAC 地址，它会先去查询自己的 ARP 高速缓存中有没有主机 B 的相关记录；

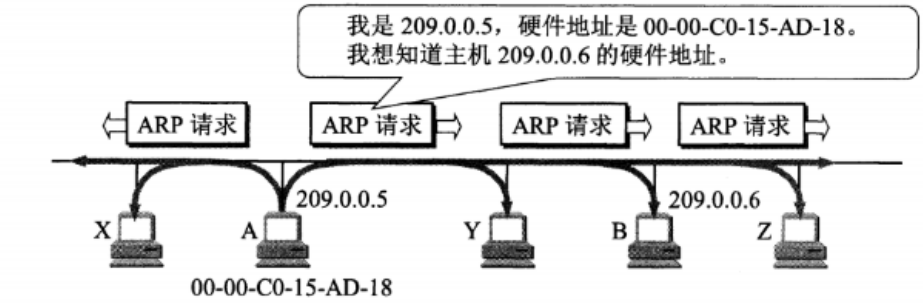
2）如果主机 A 的 ARP 高速缓存中没有主机 B 的 IP 地址到 MAC 地址的映射，主机 A 就会通过**广播**的方式发送 **ARP 请求包**（该包携带自己的 IP 地址 和 MAC 地址 以及 目标主机的 IP 地址），表明自己想要获得主机 B 的 MAC 地址；

2） 由于广播请求可以被同一个链路上的所有主机或路由器接收，因此如果这条链路上某个主机或路由的 IP 地址与这个 ARP 请求包中包含的目标主机的 IP 地址相同，那么这个节点就将自己的 MAC 地址塞入 **ARP 响应包**中返回给主机 A；

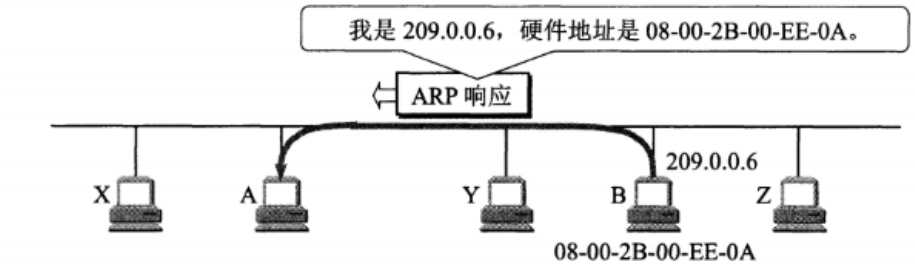
0	15	16	31（比特）
硬件类型		硬件类型	
HLEN（MAC 地址长度）	PLEN（IP 地址长度）	操作码	
发送端 MAC 地址			
发送端 MAC 地址（续）		发送端 IP 地址	
发送端 IP 地址（续）		接收端 MAC 地址	
接收端 MAC 地址（续）			
接收端 IP 地址			

> 当然，ARP 响应包是以单播的形式进行发送的，毕竟 ARP 请求包中已经包含了主机 A 的 IP 地址，所以主机 B 非常清楚这个响应包应该发送给谁。
>
> 大部分网络协议在设计的时候，都是保持极度克制的，不需要的交互就砍掉，能合并的信息就合并，能不用广播就用单播，以此让带宽变得更多让网络变得更快。

3）主机 A 在收到主机 B 发过来的 ARP 响应包后，向其 ARP 高速缓存中写入主机 B 的 IP 地址到 MAC 地址的映射。



(a) 主机 A 广播发送 ARP 请求分组



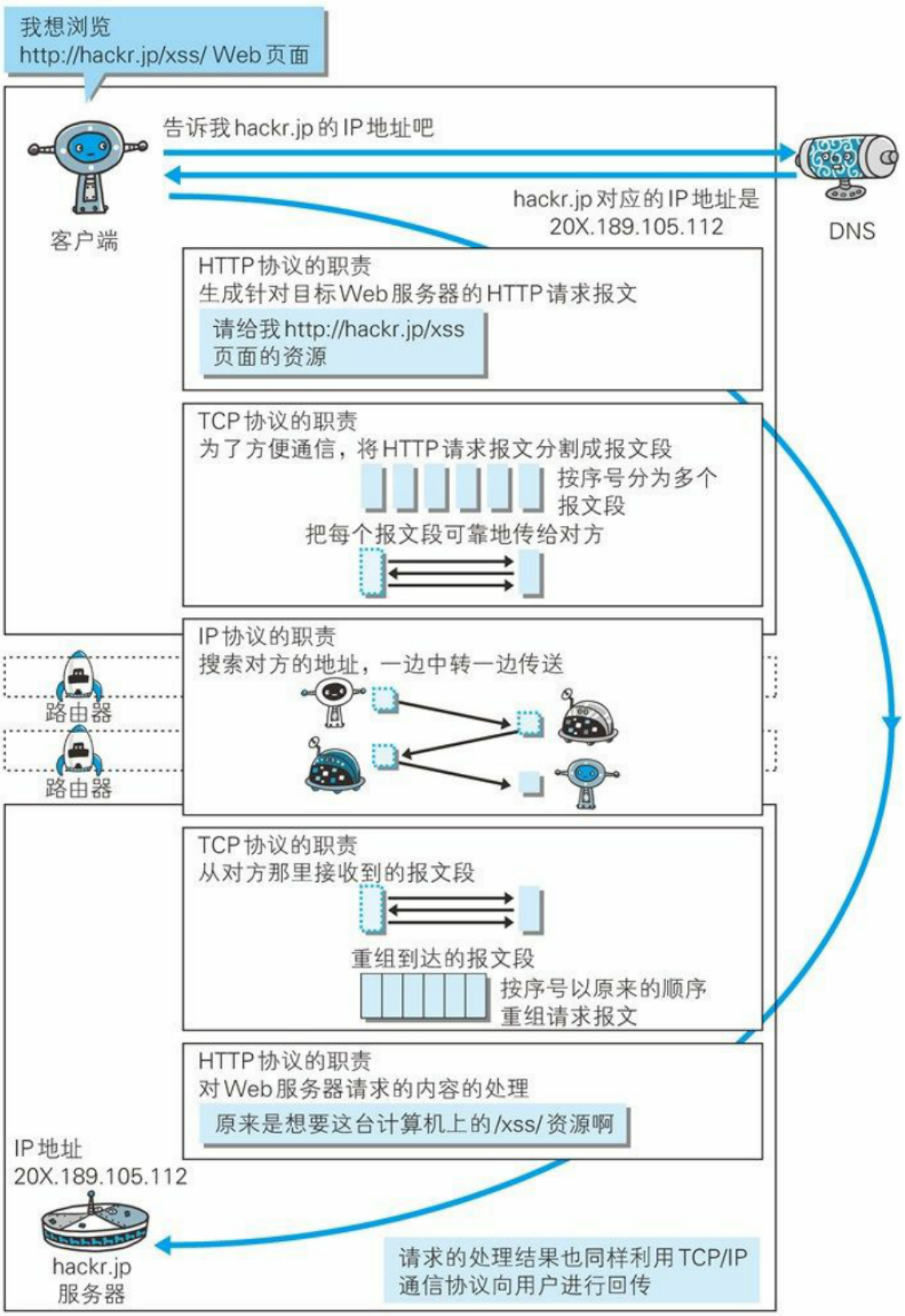
(b) 主机 B 向 A 发送 ARP 响应分组

当然，缓存是有一定期限的，超过这个期限，缓存的内容将被清空。这也使得即使 MAC 地址和 IP 地址的映射关系发生了变化，也依然能够正确的将数据包发送给目标地址。

8. 服务器响应请求

浏览器的 HTTP 请求报文通过 TCP 三次握手建立的连接通道被切分成若干报文段分别发送给服务器，服务器在收到这些报文段后，按照序号以原来的顺序重组 HTTP 请求报文。然后处理并返回一个 HTTP 响应。当然，HTTP 响应报文也要经过和 HTTP 请求报文一样的过程。

看下方这个图回顾一下（图片来源《图解 HTTP》）：



9. 断开 TCP 连接

浏览器和服务器都不再需要发送数据后，四次挥手断开 TCP 连接，详细可见 关于 TCP 三次握手和四次挥手，满分回答在此 这篇文章。

10. 浏览器显示界面

浏览器接收到服务器返回的数据包，根据浏览器的渲染机制对相应的数据进行渲染

总结

屏蔽掉底层细节，笼统的总结一下上述过程：

应用层：

- 浏览器封装 HTTP 请求报文
- DNS 解析域名获得目标服务器地址

传输层：

- 建立连接
- 把应用层传过来的 HTTP 请求报文进行分割，并在各个报文上打上标记序号及端口号转发给网络层

网络层：

- 利用 ARP 协议根据 IP 地址获取作为通信目的地的 MAC 地址后转发给链路层

服务端在链路层收到数据，按序往上层发送，一直到应用层接收到浏览器发送来的 HTTP 请求报文，然后处理该请求并返回 HTTP 响应报文，浏览器接收到响应报文之后解析渲染界面。最后 TCP 断开连接。

聊聊：生产Nginx现大量TIME-WAIT，导致连接耗尽，该如何处理?

背景说明:

在尼恩读者50+交流群中，是不是有小伙伴问：

> 尼恩，生产环境 Nginx 后端服务大量 TIME-WAIT， 该怎么办?

除了Nginx进程之外，还有其他的后端服务如：

> 尼恩，生产环境 Netty、SpringCloud Gateway 后端服务大量 TIME-WAIT， 该怎么办?

遇到这样的生产环境难题，小伙伴们非常头疼。

更为头疼的是，这个也是一道场景的面试题。之前有小伙伴反应过，他面试科大讯飞的时候，遇到了这道题目：

> 生产环境 Nginx 后端服务大量 TIME-WAIT 的解决步骤

这里尼恩给大家做一下系统化、体系化的梳理，使得大家可以充分展示一下大家雄厚的 “技术肌肉”，**让面试官爱到 “不能自已、口水直流”。**

也一并把这个题目以及参考答案，收入咱们的《尼恩Java面试宝典》，供后面的小伙伴参考，提升大家的 3高 架构、设计、开发水平。

> 注：本文以 PDF 持续更新，最新尼恩 架构笔记、面试题 的PDF文件，请从这里获取：码云

基础知识:

TIME_WAIT是什么?

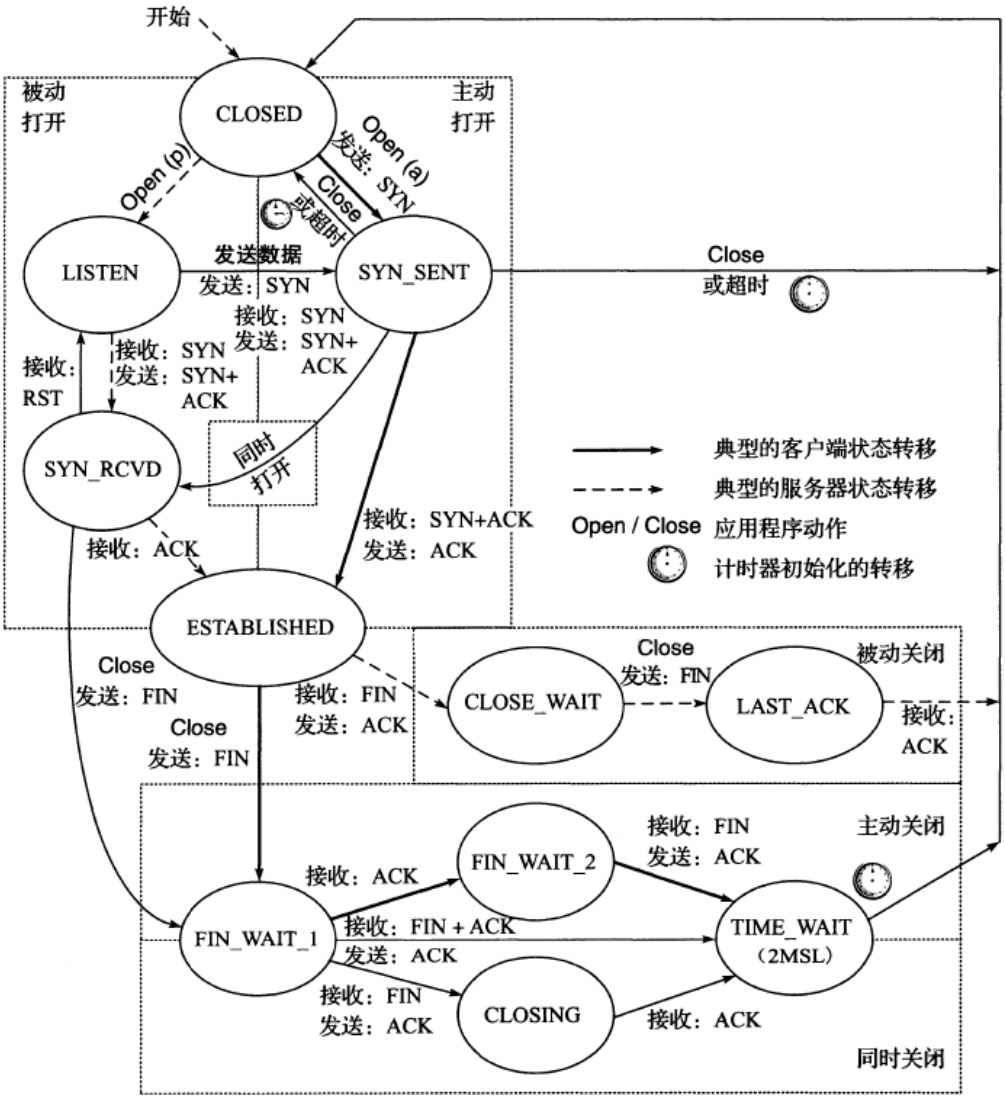


图 13-8　TCP 状态转换图（也称作有限状态机）。箭头表示因报文段传输、接收以及计时器超时而引发的状态转换。粗箭头表示典型的客户端的行为，虚线箭头表示典型的服务器行为。粗体指令（例如 open、close）是应用程序执行的操作

```
Client                                          Server
   ======                                          ======
   ESTABLISHED                                     ESTABLISHED
   (client closes)
   ESTABLISHED                                     ESTABLISHED
                -------->>
   FIN_WAIT_1
               <<---------
   FIN_WAIT_2                                      CLOSE_WAIT
               <<---------   (server closes)
                                                   LAST_ACK
               , ------->>
   TIME_WAIT                                       CLOSED
   (2*msl elapses...)
   CLOSED
```

在构建TCP客户端服务器系统时，很容易犯简单的错误，这些错误会严重限制可伸缩性。

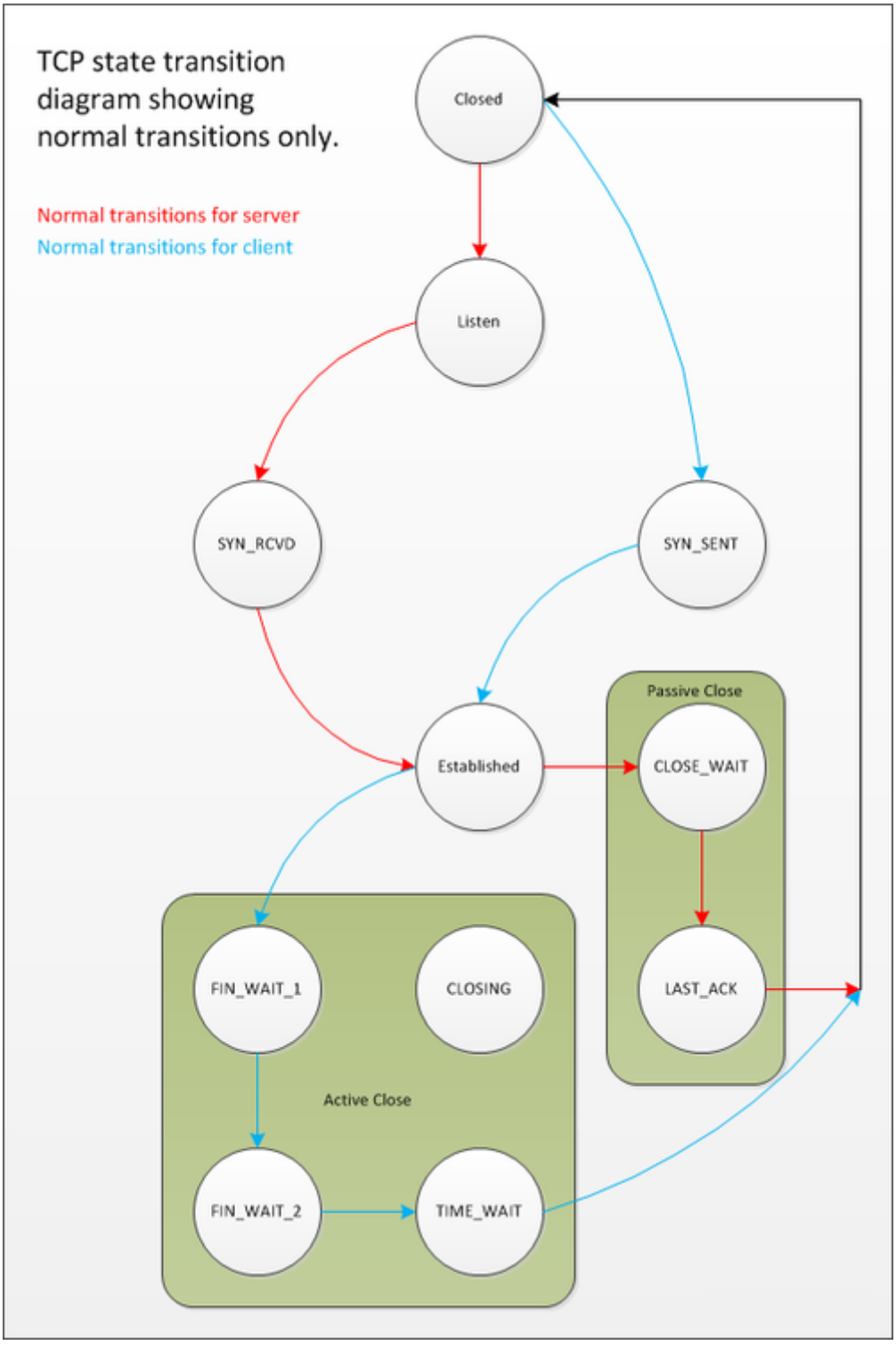
其中一个错误是没有考虑到`TIME_WAIT`状态。

为什么`TIME_WAIT`存在，它可能导致的问题，如何解决它，以及什么时候不应该。

`TIME_WAIT`是 TCP 状态转换图中经常被误解的状态。

这是某些套接字可以进入并保持相对较长时间的状态，如果您有足够的套接字，那么您创建新套接字连接的能力可能会受到影响，这可能会影响客户端服务器系统的可伸缩性。

关于套接字如何以及为什么首先进入`TIME_WAIT`，经常存在一些误解，不应该有，这并不神奇。从下面的TCP状态转换图中可以看出`TIME_WAIT`，是TCP客户端通常最终处于的最终状态。



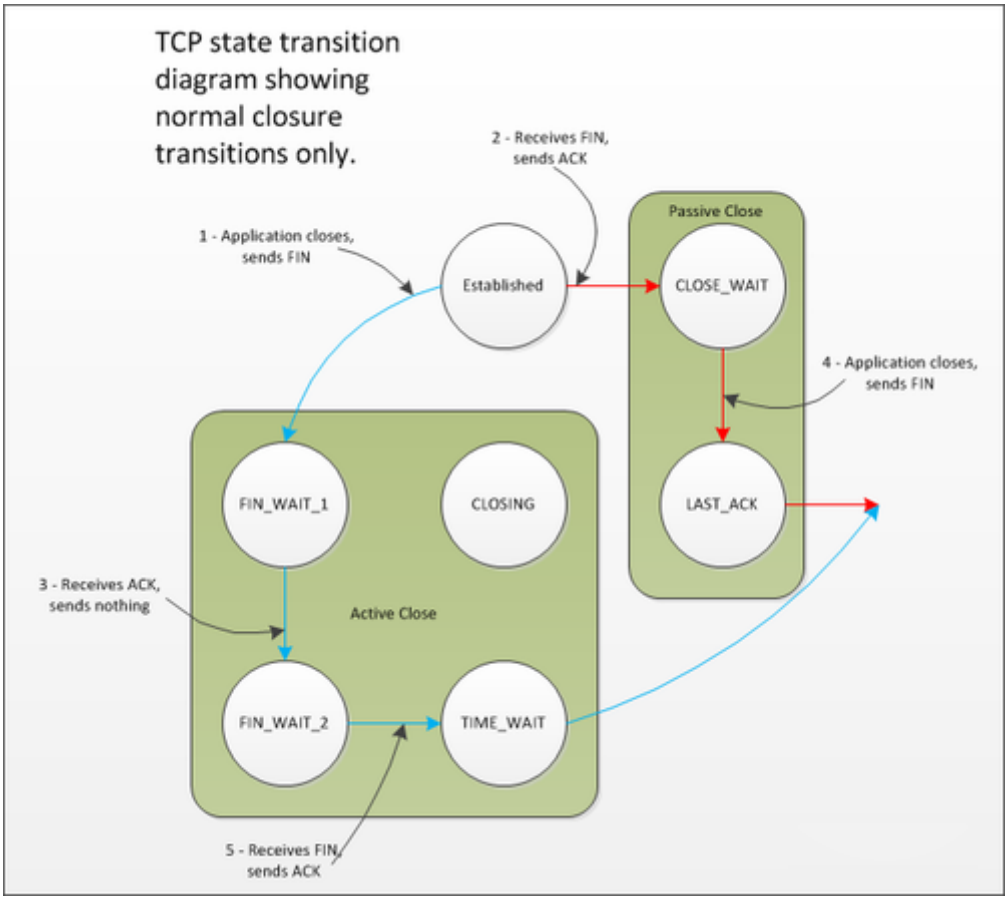
虽然状态转换图显示`TIME_WAIT`为客户端的最终状态，但它不一定是客户端`TIME_WAIT`。事实上，最终状态是启动"主动关闭"的对等端最终进入，这可以是客户端或服务器。那么，发布"主动关闭"是什么意思?

如果TCP对等方是第一个呼叫`close()`连接的对等方，则TCP对等方会启动"主动关闭"。

在许多协议和客户端/服务器设计中，这是客户端。

在HTTP和FTP服务器中，这通常是服务器。

导致同伴结束的事件的实际顺序`TIME_WAIT`如下。



现在我们知道套接字是如何结束`TIME_WAIT`的，理解为什么这个状态存在以及它为什么会成为一个潜在的问题是有用的。

`TIME_WAIT`通常也被称为2MSL等待状态。

这是因为转换到的套接字在此期间的持续时间`TIME_WAIT`为*2 x最大段寿命*。

MSL是任何段的最大时间量，对于构成TCP协议一部分的数据报的所有意图和目的而言，在丢弃之前，网络可以在网络上保持有效。这个时间限制最终以用于传输TCP段的IP数据报中的TTL字段为界。

不同的实现为MSL选择不同的值，常用值为30秒，1分钟或2分钟。

RFC 793指定MSL为2分钟，Windows系统默认为此值，但可以使用**TcpTimedWaitDelay**注册表设置进行调整。

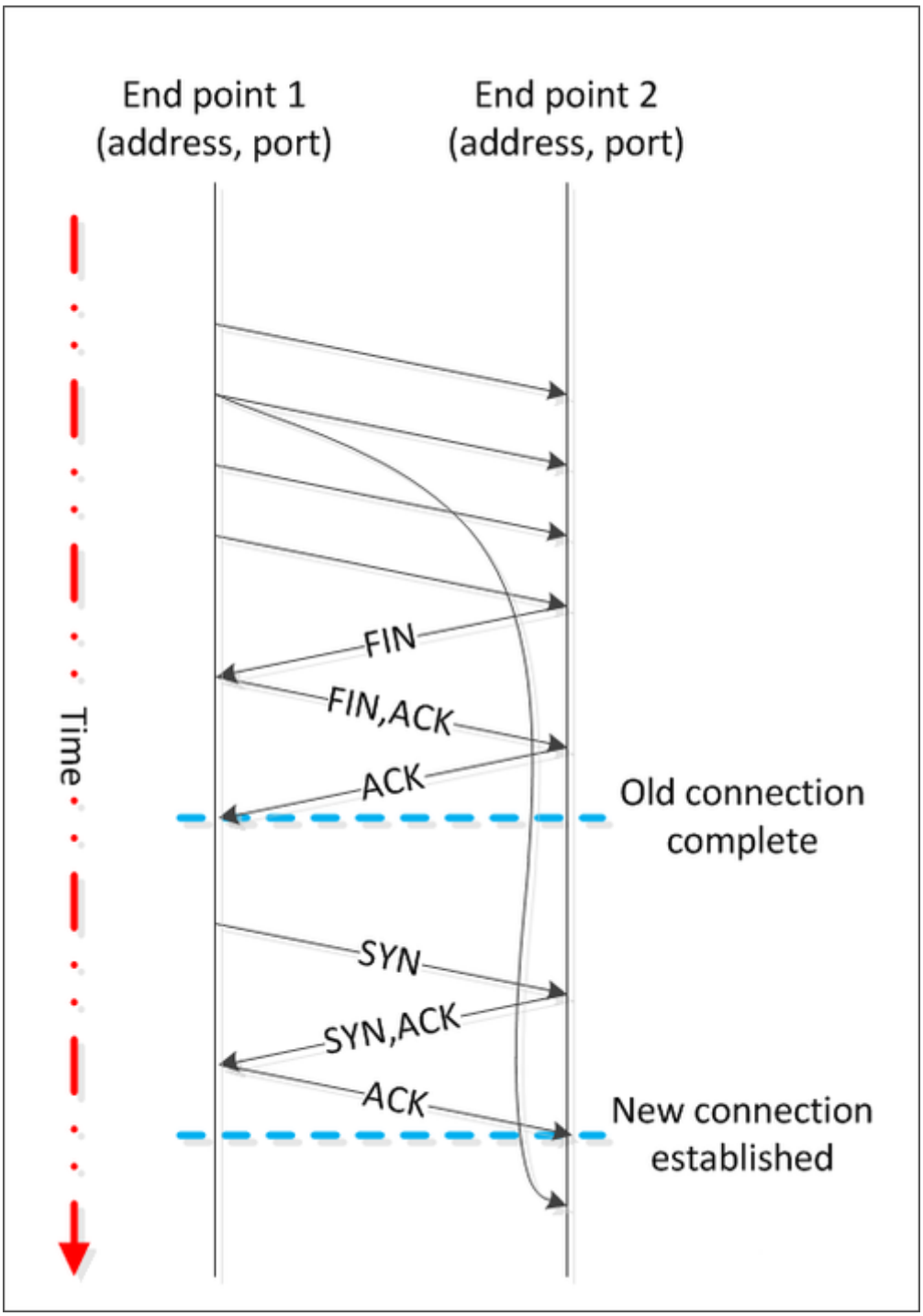
`TIME_WAIT`可能影响系统可伸缩性 的原因是，TCP连接中一个完全关闭的套接字将保持`TIME_WAIT`约4分钟的状态。

如果许多连接正在快速打开和关闭，那么套接字`TIME_WAIT`可能会开始累积在系统上; 您可以`TIME_WAIT`使用**netstat**查看套接字。一次可以建立有限数量的套接字连接，限制此数量的其中一个因素是可用本地端口的数量。

如果`TIME_WAIT`插入太多的套接字，您会发现很难建立新的出站连接，因为缺少可用于新连接的本地端口。

但为什么`TIME_WAIT`存在呢？有两个原因需要`TIME_WAIT`。

首先是防止一个连接的延迟段被误解为后续连接的一部分。丢弃在连接处于2MSL等待状态时到达的任何段。



在上图中，我们有两个从终点1到终点2的连接。每个终点的地址和端口在每个连接中都是相同的。第一个连接终止于由端点2启动的主动关闭。如果端点2没有保持`TIME_WAIT`足够长的时间以确保来自先前连接的所有分段已经失效，则延迟的分段（具有适当的序列号）可以是误认为是第二个连接的一部分...

请注意，延迟片段*很*可能会导致这样的问题。首先，每个端点的地址和端口需要相同; 这通常不太可能，因为操作系统通常从短暂端口范围为您选择客户端端口，从而在连接之间进行更改。其次，延迟段的序列号需要在新连接中有效，这也是不太可能的。但是，如果出现这两种情况，`TIME_WAIT`则会阻止新连接的数据被破坏。

需要`TIME_WAIT`**第二个原因**是可靠地实施TCP的全双工连接终端。

如果`ACK`从终点2开始的终点被丢弃，则终点1将重新发送终点`FIN`。如果连接已经过渡到`CLOSED`终点2，那么唯一可能的应答是发送一个，`RST`因为重发`FIN`是意外的。即使所有数据传输正确，这也会导致终点1接收到错误。

不幸的是，一些操作系统实现的方式`TIME_WAIT`似乎有点幼稚。

只有`TIME_WAIT`通过阻塞才能提供保护的连接与需要的socket完全匹配`TIME_WAIT`。这意味着由客户端地址，客户端端口，服务器地址和服务器端口标识的连接。但是，某些操作系统会施加更严格的限制，并且阻止本地端口号被重新使用，而该端口号包含在连接中`TIME_WAIT`。如果有足够的套接字结束，`TIME_WAIT`则不能建立新的出站连接，因为没有剩余本地端口分配给新连接。

Windows不会执行此操作，只会阻止与其中的连接*完全*匹配的出站连接`TIME_WAIT`。

入站连接受影响较小`TIME_WAIT`。

虽然由服务器主动关闭的`TIME_WAIT`连接与客户端连接完全一样，但是服务器正在侦听的本地端口不会阻止其成为新的入站连接的一部分。在Windows上，服务器正在监听的众所周知的端口可以构成随后接受的连接的一部分，并且如果从远程地址和端口建立了新的连接，该连接当前构成`TIME_WAIT`该本地地址和端口的连接的一部分，则只要新序列号大于当前连接的最终序列号，就允许连接`TIME_WAIT`。然而，`TIME_WAIT`在服务器上累积可能会影响性能和资源使用，因为`TIME_WAIT`最终需要超时的连接需要进行一些工作，直到`TIME_WAIT`状态结束，连接仍占用（少量）服务器资源。

鉴于`TIME_WAIT`由于**本地端口号耗尽而影响出站连接的建立**，并且这些连接通常使用由临时端口范围由操作系统自动分配的本地端口，因此您可以做的第一件事情是确保你正在使用一个体面的短暂端口范围。在Windows上，您可以通过调整`MaxUserPort`注册表设置来完成此操作。请注意，默认情况下，许多Windows系统的临时端口范围大约为4000，这对于许多客户端服务器系统来说可能太低。

虽然有可能缩短套接字在`TIME_WAIT`这方面的花费时间通常不会有帮助。鉴于`TIME_WAIT`当许多连接建立并且主动关闭时，这只是一个问题，调整2MSL等待周期通常会导致在给定时间内建立和关闭更多连接的情况，因此必须不断调整2MSL直到由于延迟片段似乎是后来连接的一部分，因此它可能会开始出现问题; 如果您连接到相同的远程地址和端口，并且非常快速地使用所有本地端口范围，或者如果连接到相同的远程地址和端口并将本地端口绑定到固定值，这种情况才会变得可能。

更改2MSL延迟通常是机器范围内的配置更改。您可以尝试`TIME_WAIT`使用`SO_REUSEADDR`套接字选项解决套接字级别问题。这允许在具有相同地址和端口的现有套接字已经存在的同时创建套接字。新的套接字基本上劫持了旧的套接字。您可以使用它`SO_REUSEADDR`来允许创建套接字，同时具有相同端口的套接字已经在其中，`TIME_WAIT`但这也会导致诸如拒绝服务攻击或数据窃取等问题。在Windows平台上另一套接字选项，`SO_EXCLUSIVEADDRUSE`可以帮助防止某些缺点的`SO_REUSEADDR`，但在我看来，最好避免这些尝试在各地工作`TIME_WAIT`，而是设计系统，使`TIME_WAIT`不是问题。

上面的TCP状态转换图都显示有序的连接终止。还有另一种方法来终止TCP连接，这是通过中止连接并发送一个`RST`而不是一个`FIN`。这通常通过将`SO_LINGER`套接字选项设置为0 来实现。这会导致挂起的数据被丢弃，并且连接将被中止，`RST`而不是等待传输的数据，并且使用a清除干净的连接`FIN`。重要的是要认识到，当连接被中止时，可能在对等体之间流动的任何数据将被丢弃，`RST`直接送达; 通常作为表示“连接已被同级重置”的事实的错误。远程节点知道连接被中止，并且两个节点都没有进入`TIME_WAIT`。

当然，已经中止使用的连接的新化身`RST`可能成为`TIME_WAIT`阻碍的延迟段问题的受害者，但是成为问题所需的条件是不太可能的，无论如何，参见上面的更多细节。为了防止已经中止的连接导致延迟段问题，两个对等端都必须转换到`TIME_WAIT`连接关闭可能由诸如路由器之类的中介引起。但是，这不会发生，连接的两端都会关闭。

有几件事你可以做，以避免`TIME_WAIT`成为你的问题。其中一些假定您有能力更改您的客户端和服务器之间所说的协议，但是对于自定义服务器设计，您通常会这样做。

对于永远不会建立自己的出站连接的服务器，除了维护连接的资源和性能意义之外`TIME_WAIT`，您不必过分担心。

对于确实建立出站连接并接受入站连接的服务器，黄金法则是始终确保如果`TIME_WAIT`需要发生这种情况，则它会在另一个对等而不是服务器上结束。做到这一点的最佳方式是*永远不要*从服务器发起主动关闭，无论原因是什么。如果您的对等方超时，则以中止连接`RST`而不是关闭连接。如果你的对方发送无效数据，中止连接等等。

这个想法是，如果你的服务器永远不会启动一个主动关闭，它永远不会累积`TIME_WAIT`套接字，因此永远不会受到它们导致的可伸缩性问题的困扰。虽然在出现错误情况时很容易看到如何中止连接，但正常连接终止的情况如何？理想情况下，您应该为协议设计一种方式，让服务器告诉客户端它应该断开连接，而不是简单地让服务器发起主动关闭。因此，如果服务器需要终止连接，服务器会发送一个应用程序级别"我们已经完成"的消息，客户将此消息作为关闭连接的原因。如果客户端未能在合理的时间内关闭连接，则服务器会中止连接。

在客户端上，事情稍微复杂一些，毕竟，有人必须发起一个主动关闭来干净地终止TCP连接，并且如果它是客户端，那么这就是`TIME_WAIT`最终会结束的地方。然而，`TIME_WAIT`最终在客户端有几个优点。

首先，如果由于某种原因，客户端由于套接字在`TIME_WAIT`一个客户端中的积累而最终导致连接问题。其他客户不会受到影响。

其次，快速打开和关闭TCP连接到同一台服务器是没有效率的，因此超出了问题的意义`TIME_WAIT`尝试维持更长时间的连接而不是更短的时间。不要设计一个协议，客户端每分钟连接一次，并打开一个新的连接。而是使用持久连接设计，并且只有在连接失败时才重新连接，如果中间路由器拒绝在没有数据流的情况下保持连接处于打开状态，则可以实现应用程序级别ping，使用TCP保持活动状态或仅接受路由器重置连接; 好处是你没有积累`TIME_WAIT`插座。如果你在连接上做的工作自然是短暂的，那么考虑某种形式的"连接池"设计，从而保持连接的开放性和重用性。

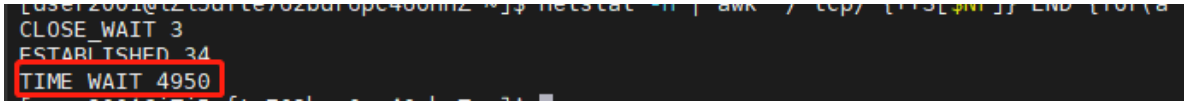
最后，如果您绝对必须快速地从客户端打开和关闭连接到同一台服务器，那么也许您可以设计一个可以使用的应用程序级别关闭顺序，然后按照此顺序关闭。您的客户可能会发送"我完成了"消息，您的服务器可能会发送"再见"消息，然后客户端可能会中止连接。

`TIME_WAIT`存在原因并通过缩短2MSL周期或允许地址重用来解决这个问题`SO_REUSEADDR`并不总是一个好主意。

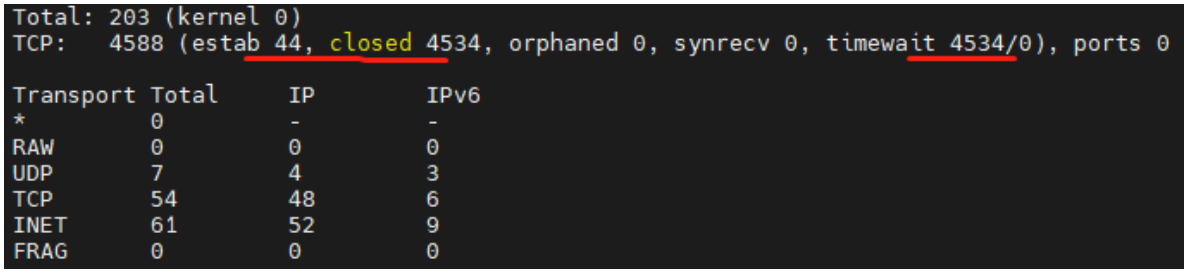
如果你能够设计你的协议`TIME_WAIT`避免在脑海中，那么你通常可以完全避免这个问题。

TIME_WAIT的4种查询方式

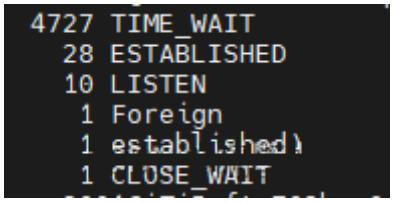
1、netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'



2、ss -s

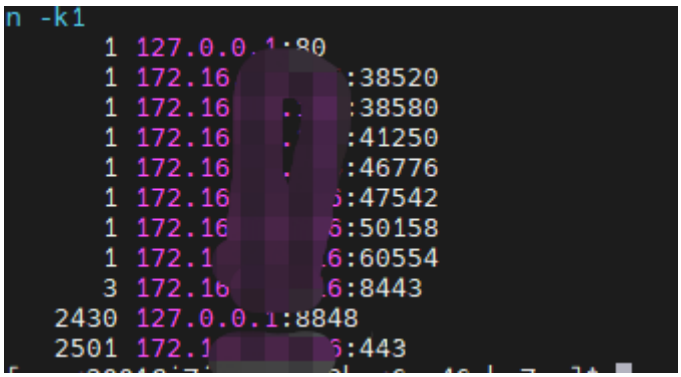


3、netstat -nat |awk '{print $6}'|sort|uniq -c|sort -rn



4、 统计TIME_WAIT 连接的本地地址

netstat -an | grep TIME_WAIT | awk '{print $4}' | sort | uniq -c | sort -n -k1



尝试抓取 tcp 包

```
tcpdump tcp -i any -nn port 8080 | grep "172.11.11.11"
```

系统参数优化

处理方法：需要修改内核参数开启重启：

```
net.ipv4.tcp_tw_reuse = 1，开启快速回收
net.ipv4.tcp_tw_recycle = 1 （在NAT网络中不建议开启，要设置为0），并且开启
net.ipv4.tcp_timestamps = 1以上两个参数才生产。
```

具体操作方法如下：

```
echo "net.ipv4.tcp_tw_reuse = 1" >> /etc/sysctl.conf;
echo "net.ipv4.tcp_tw_recycle = 1" >> /etc/sysctl.conf;
echo "net.ipv4.tcp_timestamps = 1" >> /etc/sysctl.conf;
```

sysctl -p;

nginx 配置优化

当使用nginx作为反向代理时，为了支持长连接，需要做到两点：

1. 从client到nginx的连接是长连接
2. 从nginx到server的连接是长连接

保持和client的长连接：

```
http {
    keepalive_timeout 120s 120s;
    keepalive_requests 1000;
}
```

keepalive_timeout设置

语法

```
keepalive_timeout timeout [header_timeout];
```

第一个参数：设置keep-alive客户端(浏览器)连接在服务器端(nginx端)保持开启的超时值（默认75s）；值为0会禁用keep-alive客户端连接；

第二个参数：可选、在响应的header域中设置一个值“Keep-Alive: timeout=time”；通常可以不用设置；

这个keepalive_timeout针对的是浏览器和nginx建立的一个tcp通道，没有数据传输时最长等待该时候后就关闭.

nginx和upstream中的keepalive_timeout则受到tomcat连接器的控制，tomcat中也有一个类似的keepalive_timeout参数

keepalive_requests理解设置

keepalive_requests指令用于设置一个keep-alive连接上可以服务的请求的最大数量，当最大请求数量达到时，连接被关闭。默认是100。

这个参数的真实含义，是指一个keep alive建立之后，nginx就会为这个连接设置一个计数器，记录这个keep alive的长连接上已经接收并处理的客户端请求的数量。如果达到这个参数设置的最大值时，则nginx会强行关闭这个长连接，逼迫客户端不得不重新建立新的长连接。

保持和server的长连接

为了让nginx和后端server（nginx称为upstream）之间保持长连接，典型设置如下：（默认nginx访问后端都是用的短连接(HTTP1.0)，一个请求来了，Nginx 新开一个端口和后端建立连接，后端执行完毕后主动关闭该链接）

```
location / {
    proxy_http_version 1.1;
    proxy_set_header Connection keep-alive;
    proxy_pass http://httpurl;
}
```

HTTP协议中对长连接的支持是从1.1版本之后才有的，因此最好通过proxy_http_version指令设置为”1.1”；

从client过来的http header，因为即使是client和nginx之间是短连接，nginx和upstream之间也是可以开启长连接的。

keepalive理解设置

此处keepalive的含义不是开启、关闭长连接的开关；也不是用来设置超时的timeout；更不是设置长连接池最大连接数。官方解释：

1. The connections parameter sets the maximum number of idle keepalive connections to upstream servers connections（设置到upstream服务器的空闲keepalive连接的最大数量）
2. When this number is exceeded, the least recently used connections are closed. （当这个数量被突破时，最近使用最少的连接将被关闭）
3. It should be particularly noted that the keepalive directive does not limit the total number of connections to upstream servers that an nginx worker process can open.（特别提醒：keepalive指令不会限制一个nginx worker进程到upstream服务器连接的总数量）

总结：

keepalive 这个参数一定要小心设置，尤其对于QPS比较高的场景，推荐先做一下估算，根据QPS和平均响应时间大体能计算出需要的长连接的数量。

比如前面10000 QPS和100毫秒响应时间就可以推算出需要的长连接数量大概是1000. 然后将keepalive设置为这个长连接数量的10%到30%。比较懒的同学，可以直接设置为keepalive=1000之类的，一般都OK的了。

综上，出现大量`TIME_WAIT`的情况

1）导致 nginx端出现大量`TIME_WAIT`的情况有两种：

keepalive_requests设置比较小，高并发下超过此值后nginx会强制关闭和客户端保持的keepalive长连接；（主动关闭连接后导致nginx出现`TIME_WAIT`）

keepalive设置的比较小（空闲数太小），导致高并发下nginx会频繁出现连接数震荡（超过该值会关闭连接），不停的关闭、开启和后端server保持的keepalive长连接；

2）导致后端server端出现大量`TIME_WAIT`的情况：

nginx没有打开和后端的长连接，即：没有设置`proxy_http_version 1.1;`和`proxy_set_header Connection “”;`从而导致后端server每次关闭连接，高并发下就会出现server端出现大量`TIME_WAIT`

推荐阅读：

《网易二面：CPU狂飙900%，该怎么处理？》

《阿里二面：千万级、亿级数据，如何性能优化？ 教科书级 答案来了》

《峰值21WQps、亿级DAU，小游戏《羊了个羊》是怎么架构的？》

《场景题：假设10W人突访，你的系统如何做到不 雪崩？》

《2个大厂 100亿级 超大流量 红包 架构方案》

《Nginx面试题（史上最全 + 持续更新）》

《K8S面试题（史上最全 + 持续更新）》

《操作系统面试题（史上最全、持续更新）》

《Docker面试题（史上最全 + 持续更新）》

《Springcloud gateway 底层原理、核心实战 (史上最全)》

《Flux、Mono、Reactor 实战（史上最全）》

《sentinel （史上最全）》

《Nacos (史上最全)》

《TCP协议详解 (史上最全)》

《分库分表 Sharding-JDBC 底层原理、核心实战（史上最全）》

《clickhouse 超底层原理 + 高可用实操 （史上最全）》

《nacos高可用（图解+秒懂+史上最全）》

《队列之王： Disruptor 原理、架构、源码 一文穿透》

《环形队列、 条带环形队列 Striped-RingBuffer （史上最全）》

《一文搞定：SpringBoot、SLF4j、Log4j、Logback、Netty之间混乱关系（史上最全）》

《单例模式（史上最全）》

《红黑树（ 图解 + 秒懂 + 史上最全）》

《分布式事务 （秒懂）》

《缓存之王：Caffeine 源码、架构、原理（史上最全，10W字 超级长文）》

《缓存之王：Caffeine 的使用（史上最全）》

《Java Agent 探针、字节码增强 ByteBuddy（史上最全）》

《Docker原理（图解+秒懂+史上最全）》

《Redis分布式锁（图解 - 秒懂 - 史上最全）》

《Zookeeper 分布式锁 - 图解 - 秒懂》

《Zookeeper Curator 事件监听 - 10分钟看懂》

《Netty 粘包 拆包 | 史上最全解读》

《Netty 100万级高并发服务器配置》

《Springcloud 高并发 配置 （一文全懂）》

硬核推荐：尼恩Java硬核架构班

又名疯狂创客圈社群 VIP

详情：https://www.cnblogs.com/crazymakercircle/p/9904544.html


规划中
《架构师实操篇：基于netty 手写 rpc 框架- 参考 dubbo、seata rpc框架》
《架构师实操篇：go语言学习，以及基于 go 手写 rpc 框架》
《架构师实操篇：千万级任务调度平台 架构与实操- 基于尼恩17年的亿级搜索项目》
《架构师实操篇：工业级 亿级文档搜索 平台 架构与实操- 基于尼恩17年的亿级搜索项目》
特色
会员制
提供技术方向指导，
职业生涯指导，少躺坑，少弯路
简历指导
这个很重要，
对于挪窝涨薪来说
实操性
以上项目，都是老架构师
在生产上实操过的项目
非水货
40岁老架构师，不是水货架构师
《Java高并发三部曲》为证

架构班（社群 VIP）的起源：

最初的视频，主要是给读者加餐。很多的读者，需要一些高质量的实操、理论视频，所以，我就围绕书，和底层，做了几个实操、理论视频，然后效果还不错，后面就做成迭代模式了。

架构班（社群 VIP）的功能：

提供高质量实操项目整刀真枪的架构指导、快速提升大家的:

- 开发水平
- 设计水平
- 架构水平

弥补业务中 CRUD 开发短板，帮助大家尽早脱离具备 3 高能力，掌握：

- 高性能
- 高并发
- 高可用

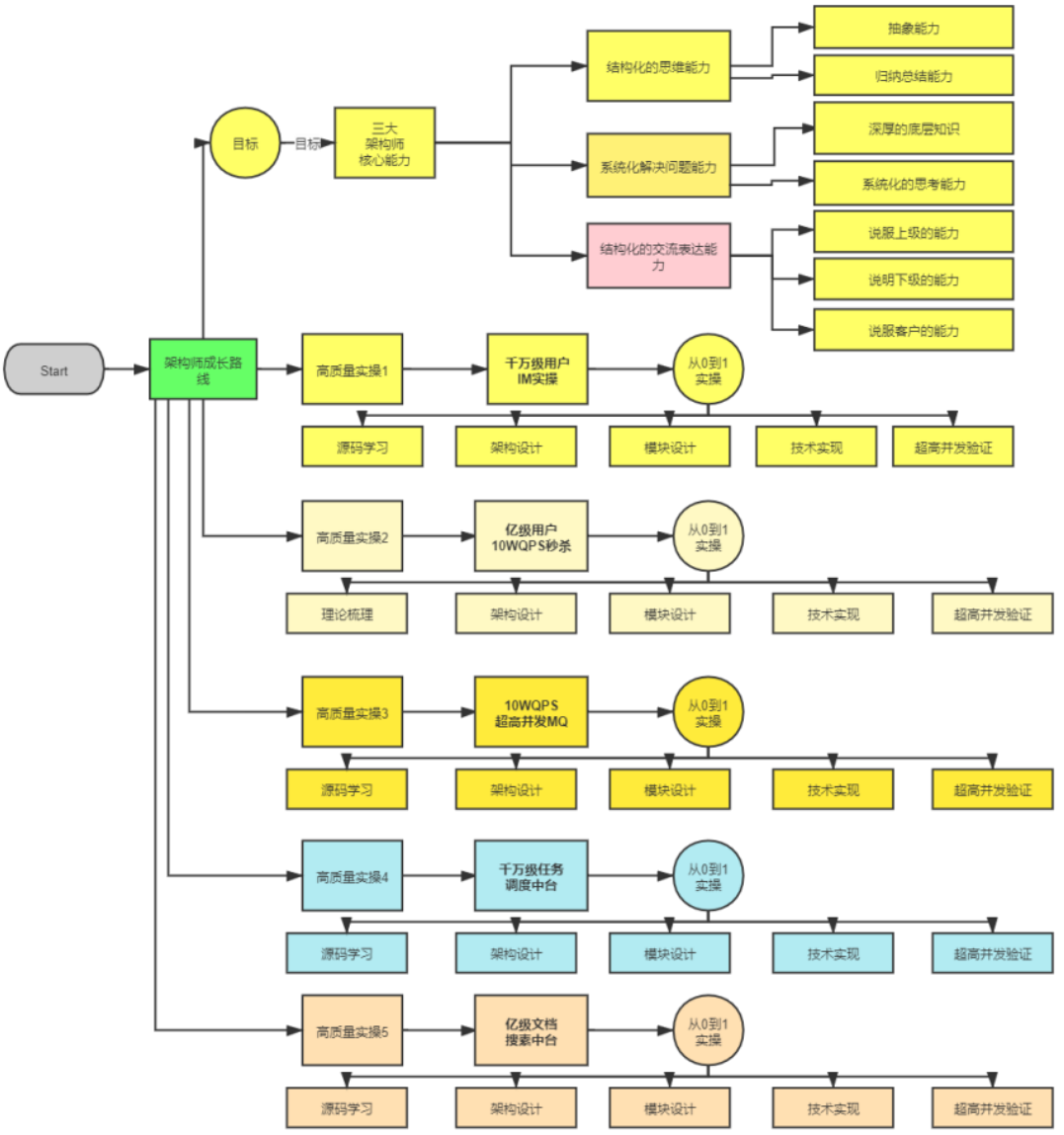
作为一个高质量的架构师成长、人脉社群，把所有的卷王聚焦起来，一起卷：

- 卷高并发实操
- 卷底层原理
- 卷架构理论、架构哲学
- 最终成为顶级架构师，实现人生理想，走向人生巅峰

架构班（社群 VIP）的目的：

- 高质量的实操，大大提升简历的含金量，吸引力，增强面试的召唤率
- 为大家提供九阳真经、葵花宝典，快速提升水平
- 进大厂、拿高薪
- 一路陪伴，提供助学视频和指导，辅导大家成为架构师
- 自学为主，和其他卷王一起，卷高并发实操，卷底层原理、卷大厂面试题，争取狠卷 3 月成高手，狠卷 3 年成为顶级架构师

N 个超高并发实操项目：简历压轴、个顶个精彩



【样章】第 17 章:横扫全网 Rocketmq 视频第 2 部曲: 工业级 rocketmq 高可用(HA)底层原理和实操

工业级 rocketmq 高可用底层原理，包含：消息消费、同步消息、异步消息、单向消息等不同消息的底层原理和源码实现；消息队列非常底层的主从复制、高可用、同步刷盘、异步刷盘等底层原理。

工业级 rocketmq 高可用底层原理和搭建实操，包含：高可用集群的搭建。

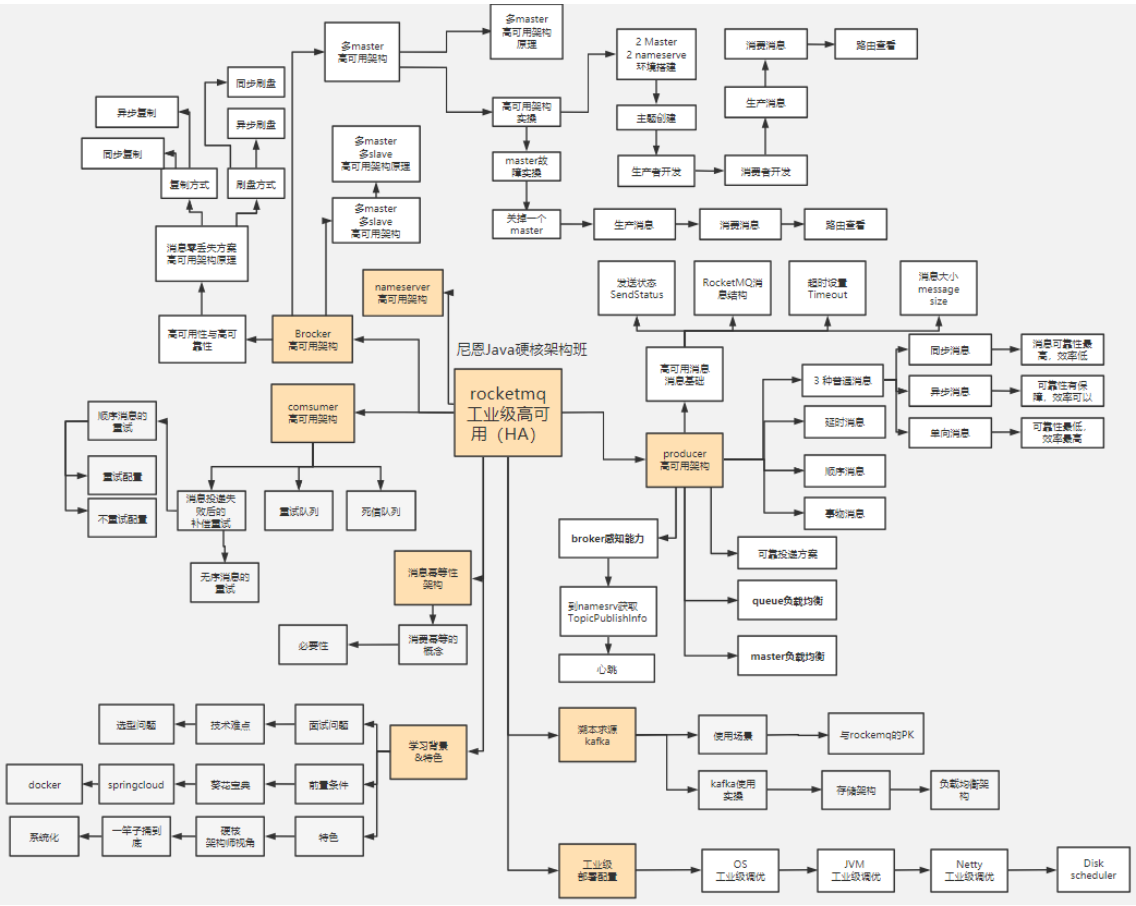
解决以下难题：

1、技术难题：RocketMQ 如何最大限度的保证消息不丢失的呢？RocketMQ 消息如何做到高可靠投递？

2、技术难题：基于消息的分布式事务，核心原理不理解

3、选型难题： kafka or rocketmq ，该娶谁？

下图链接：https://www.processon.com/view/6178e8ae0e3e7416bde9da19



成功案例：2 年翻 3 倍，35 岁卷王成功转型为架构师

详情：http://topcoder.cloud/forum.php?mod=forumdisplay&fid=43&page=1

最新　最后发表　热门　精华

成功案例：[1057号卷王] 3年小伙拿到外企offer，薪酬涨了200%
卷王1号　超级版主　前天 17:41

成功案例：[645号卷王] 4年经验卷王逆袭，被毕业后，反涨24W
卷王1号　超级版主　2022-9-21

成功案例：[878号卷王] 小伙8年经验，年薪60W
卷王1号　超级版主　2022-8-13

年薪70W案例：通过尼恩的指导，小伙伴年薪从40W涨到70W
卷王1号　超级版主　2022-2-11

成功案例：[493号卷王] 5年小伙拿满意offer，就业寒冬季逆涨30%
卷王1号　超级版主　前天 17:43

成功案例：[250号卷王] 就业极寒时代，收offer 涨25%
卷王1号　超级版主　前天 17:38

成功案例：[612号卷王] 就业极寒时代，从外包到自研
卷王1号　超级版主　前天 17:15

成功案例：[913号卷王] 热烈祝贺6年经验卷王，年薪40W
卷王1号　超级版主　2022-9-21

成功案例：[959号卷王] 4年经验卷王，喜获百度、Boss直聘等N个优质offer，最高涨100%
卷王1号　超级版主　2022-9-21

成功案例：[529号卷王] 5年经验卷王喜收2大offer，最高涨5K
卷王1号　超级版主　2022-9-21

成功案例：[811号卷王] 热烈祝贺7年经验卷王，薪酬涨30%
卷王1号　超级版主　2022-9-21

成功案例：[287号卷王] 不惧大寒潮，卷王逆市收4 offer，涨30%，可喜可贺
卷王1号　超级版主　2022-5-30

成功案例：[1002号卷王] 5月份"被毕业"，改简历后，斩获顶级央企Offer，涨薪7000+
卷王1号　超级版主　2022-7-5

成功案例：[7号卷王] 热烈祝贺小伙伴涨薪120%
卷王1号 超级版主 2022-8-13

成功案例：[134号卷王] 大三小伙卷1年，斩获顶级央企Offer，成功逆袭
卷王1号 超级版主 2022-7-6

成功案例：[1008号卷王] 5年经验卷王收42W offer，月涨8000，可喜可贺
卷王1号 超级版主 2022-5-30

成功案例：[453号卷王] 非全日制 6年卷王喜提3 offer，年薪30W，可喜可贺
卷王1号 超级版主 2022-5-21

成功案例：[924号卷王] 6年卷王喜提4 offer，最高涨薪9000，可喜可贺
卷王1号 超级版主 2022-5-21

成功案例：[15号卷王] 4年卷王入职 微软，涨薪50%，可喜可贺
卷王1号 超级版主 2022-5-12

成功案例：[527号卷王] 4年卷王喜提2 offer，涨薪50%，可喜可贺
卷王1号 超级版主 2022-5-13

成功案例：[788号卷王] 3年卷王喜提优质Offer，涨薪60%
卷王1号 超级版主 2022-5-11

成功案例：热烈祝贺：非全日制卷王，喜提2个心仪offer，面3家过2家
卷王1号 超级版主 2022-4-21

成功案例：[693号卷王] 二线城市6年卷王喜提4大优质Offer，含央企offer，最高薪酬35W
卷王1号 超级版主 2022-4-16

成功案例：[85号卷王] 双非2本小伙，春招大捷，喜提9个offer，最高薪酬近30万
卷王1号 超级版主 2022-4-14

成功案例：[741号卷王] 卷王逆袭！6年小伙从很少面试机会到搞定35K*14薪Offer
卷王1号 超级版主 2022-4-12

成功案例：[642号卷王] 热烈祝贺，6年卷王喜提优质国企offer
卷王1号 超级版主 2022-4-7

成功案例：[796号卷王] 热烈祝贺，36岁卷王喜提52万优质offer
卷王1号 超级版主 2022-3-25

成功案例：[15号卷王] 小伙卷1年，涨薪9K+，喜收ebay等多个优质offer

卷王1号 超级版主 2022-3-24

成功案例：[821号卷王] 小伙狠卷3个月，喜提10多个offer

卷王1号 超级版主 2022-3-21

成功案例：[736号卷王] 3年半经验收22k offer，但是小伙志存高远，冲击25k+

卷王1号 超级版主 2022-3-20

成功案例：热烈祝贺1群小卷王offer拿到手软，甚至拒了阿里offer

卷王1号 超级版主 2022-3-16

简历案例：简历一改，腾讯的邀请就来了！热烈祝贺，小伙收到一大堆面试邀请

卷王1号 超级版主 2022-3-10

成功案例：祝贺我圈两大超级卷王，一个过了阿里HR面，一个过了阿里2面

卷王1号 超级版主 2022-3-10

成功案例：小伙伴php转Java，卷1.5年Java，涨薪50%，喜收多个优质offer

卷王1号 超级版主 2022-3-10

成功案例：4年小伙狠卷半年，拿到 移动、京东 两大顶级offer

尼恩 超级版主 2022-3-5

成功案例：[267号卷王] 助力3年经验卷王，拿到蜂巢的17k x 14薪的offer

卷王1号 超级版主 2022-2-27

成功案例：[143号卷王] 二本院校00后卷神，毕业没到一年跳到字节，年薪45W

卷王1号 超级版主 2022-2-27

成功案例：[494号卷王] 尼恩分布式事务助力卷王拿到 中信银行offer

卷王1号 超级版主 2022-2-27

成功案例：[76号卷王] 2线城市卷王，狠卷1.5年，喜收22K offer

卷王1号 超级版主 2022-2-27

成功案例：[429号卷王] 小伙伴在社群卷5个月，涨8k+

卷王1号 超级版主 2022-2-27

成功案例：[154号卷王] 双非学校毕业卷王，连拿 京东到家&滴滴 两个大厂Offer

卷王1号 超级版主 2022-2-27

成功案例：[232号卷王] 涨薪10K，继续卷向食物链顶端

卷王1号 超级版主 2022-2-27

成功案例：狠卷1年技术，喜收 腾讯、阿里、微软三大Offer，最高年薪56W

卷王1号 超级版主 2022-2-27

成功案例：[449号卷王] 应届毕业卷王喜收 滴滴offer, 年薪33W

卷王1号 超级版主 2022-2-27

成功案例：[551号卷王] 小伙伴学完后，成功进入大厂，并且推荐自己的朋友加VIP学习

卷王1号 超级版主 2022-2-10

成功案例：[214号卷王] 助力2年经验卷王，成功拿到17K月薪

卷王1号 超级版主 2022-2-10

成功案例：[92号卷王] 课程实操助力社群小伙伴喜收 喜马拉雅Offer

卷王1号 超级版主 2022-2-10

成功案例：社群卷王小伙伴成功过了滴滴三面 获滴滴Offer

卷王1号 超级版主 2022-2-10

[612号卷王]滴滴小伙伴，蹲点考察半年，觉得靠谱后加入 疯狂创客圈

卷王1号 超级版主 2022-2-10

成功案例：[732号卷王] 尼恩助力3年经验卷王收获 京东offer, 年薪35W

卷王1号 超级版主 2022-2-27

成功案例：[558号卷王] 2年经验卷王，喜收 网易和阿里子公司两个优质offer

卷王1号 超级版主 2022-2-27

成功案例：[569号卷王] 双非应届生卷王, 喜收字节跳动实习offer

卷王1号 超级版主 2022-2-25

成功案例：[420号卷王] 狠卷1年，卷王涨薪80%，涨薪12000元!

卷王1号 超级版主 2022-2-25

成功案例：[76号卷王] 通过尼恩1年半的指导，专科学历小伙伴从0.8K涨到22K

卷王1号 超级版主 2022-2-10

简历优化后的成功涨薪案例（VIP含免费简历优化）


1.5年小伙搞定15K offer
就业寒冬涨100%
5月7日改简历
11月21日晒offer
秘诀:
简历指导+狠狠卷
多谢啦
你好，好多公司都觉得我的简历太简单了，我感觉是不是没法优化了
简历.pdf
397.4 KB
好的
晚上指导你改
推送中台需要加不
我还没做啊，准备八股文暂时没时间搞这些，入职了会开始搞
OK
还有其他的项目吗
一个项目，太少啦
那种练习项目也行
他那个不止两倍吧，我才是两倍，原先7k，现在15k
那也很牛
都翻倍了，都是牛人
正常就30%
慢慢来
下一步可以瞄准25k
其实我工作一年
那你就更牛逼拉
一年经验，搞定15k offer，舍你其谁


卷王逆袭成功案例
6年小伙从很少面试机会到
搞定35K*14薪
3月5日改简历
4月11日拿offer
面试邀请少
小伙很苦恼
理想很丰满
目标35K
我要努力争取35+
一个月拿到了
理想的offer
面试法宝
rocketmq四部曲

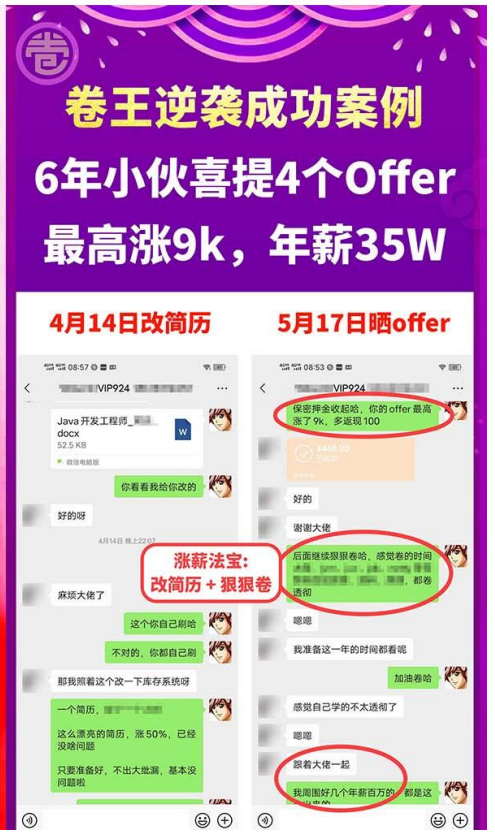

6年 经验小伙伴
喜收25K offer
3月12日改简历
12月1日晒offer
秘诀:
简历指导+狠狠卷
老师
咱们以这个项目为模板改哈
嗯好
项目有多少人，你带领多少人？
你工作几年了？
6年了
项目有多少人，你带领多少人？
语音交流一下
好
你方便吗
OK
不知道往哪个方向发展
改简历挪窝，20 涨到 25 k，高级开发，P7带的后端团队
任务重，道路远，并不是没有方向


7年经验卷王
薪酬涨30%
7月11日改简历
9月1日晒offer
秘诀:
改简历＋狠狠卷
咱们开始修改哈
要语音还是打字也行，还没到家
解决过最复杂的难题是啥？说3个哈
看看，有没有问题？
没问题
只能挪窝
下次你喊足一点哦
只要本事好，拿offer比较容易的
喊了，但是我流水太差了。。。我这家是补贴，银行不认这玩意，只能在上家的基础上加到30%封顶（特别优秀除外，我只是不错。。。）
入职了，最终并不能大幅涨薪，还是旧的太低，只涨了30%
恭喜一下
现在不比往年
能涨30%是大牛
拿到offer都算大牛
现在很多人投几百发，连个电话都没有
你已经很


4年经验卷王逆袭
被毕业后，反涨24W
7月改简历
8月30日晒offer
秘诀：
改简历＋狠狠卷

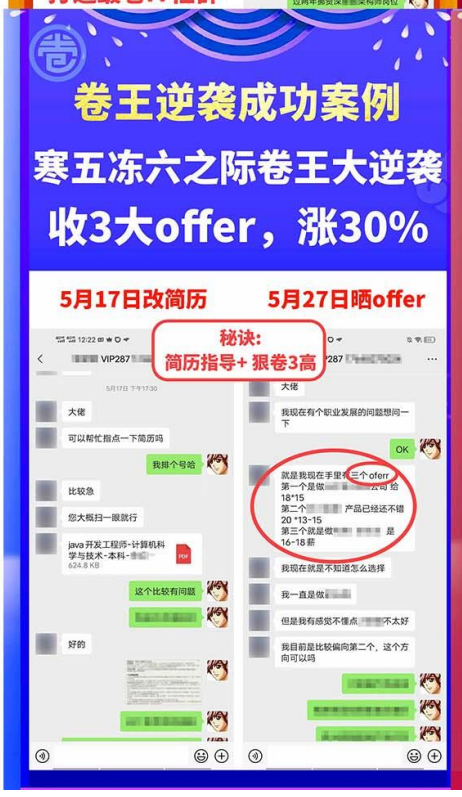

小伙5月份"被毕业"，改简历后
斩获顶级央企Offer
涨薪7000+
5月29日改简历
7月5日晒offer
秘诀：
简历指导＋狠卷3高


卷王逆袭成功案例
武汉6年喜收4个优质offer
最高的年薪35W
2月9日改简历
4月15日晒offer
面试法宝：
改简历＋实操


卷王逆袭成功案例
6年小伙喜提4个Offer
最高涨9k，年薪35W
4月14日改简历
5月17日晒offer
涨薪法宝：
改简历＋狠狠卷


卷王逆袭成功案例
5年经验小伙收2个offer
最高涨薪8k，年薪42W
5月9日改简历
5月30日晒offer
秘诀:
简历指导+ 狠卷3高
以此为样
大家狠狠卷
打造最卷IT社群


卷王逆袭成功案例
非全日制 6年经验卷王
喜提3个Offer，年包30W
5月9日改简历
5月18日晒offer
面试法宝:
改简历 + 狠狠卷


卷王逆袭成功案例
寒五冻六之际卷王大逆袭
收3大offer，涨30%
5月17日改简历
5月27日晒offer
秘诀:
简历指导+ 狠卷3高


卷王逆袭成功案例
4年卷王入职微软，涨50%
3月7日改简历
5月12日晒offer
涨薪法宝:
改简历 + 狠狠卷


4年小伙喜收百度、Boss直聘
等N个顶级Offer
最高涨幅100%
6月27日改简历
9月19日晒offer
秘诀:
改简历＋狠狠卷


卷王逆袭成功案例
4年卷王入收2个offer，涨50%
3月23日改简历
5月12日晒offer
涨薪法宝:
改简历＋狠狠卷


小伙大三暑期很焦虑
跟着尼恩卷一年
校招斩获顶级央企Offer
去年5月19日加入VIP群
今年7月5日晒offer
秘诀:
狠狠卷 书+视频


小伙高中学历
薪酬涨120%
5月6日改简历
7月22日晒offer
秘诀:
改简历＋狠狠卷


卷王逆袭成功案例
非全日制卷王 面试3家
收2个offer 涨薪30%
4月13日改简历
4月21日晒offer
面试法宝：
改简历＋面试题


5年卷王喜收2大Offer
最高涨5K
5月19日改简历
9月13日晒offer
秘诀：
改简历＋狠狠卷


卷王逆袭成功案例
3年经验卷王，涨60%
4月16日改简历
5月11日晒offer
涨薪法宝：
改简历＋狠狠卷


卷王逆袭成功案例
双非二本小伙春招大翻身
喜提9大offer
2月22日改简历
4月13日晒offer
面试法宝：
改简历＋IM实操
9大offer 最高年薪30万

修改简历找尼恩（资深简历优化专家）

- 如果面试表达不好，尼恩会提供 简历优化指导
- 如果项目没有亮点，尼恩会提供 项目亮点指导
- 如果面试表达不好，尼恩会提供 面试表达指导

作为 40 岁老架构师，尼恩长期承担技术面试官的角色：

- **从业以来，“阅历”无数，对简历有着点石成金、改头换面、脱胎换骨**的指导能力。
- **尼恩指导过刚刚就业的小白，也指导过 P8 级的老专家，都指导他们上岸。**

如何联系尼恩。尼恩微信，请参考下面的地址：

语雀：https://www.yuque.com/crazymakercircle/gkkw8s/khigna
码云：https://gitee.com/crazymaker/SimpleCrayIM/blob/master/疯狂创客圈总目录.md