**TCP和UDP的区别**

传输层协议-》连接、无连接-》有界、无界-》有序、无序-》可靠、不可靠

**1．TCP面向连接，UDP无连接**

tcp每次通信需要通过三次握手建立连接而udp不需要，因此udp传输效率更加高，tcp的连接分为长连接和短连接，TCP长连接是指再建立完成连接链路的时候，在链路空闲的时候并不结束这条线路，而是通过心跳包一直维持这条链路的连接，短连接则是每次通信结束后，连接中断，下次通信时重新建立连接，连接会产生大量的耗电，因此udp更加省电

**2．TCP无界，UDP有界；**

TCP通过字节流传输，每个TCP套接口有一个发送缓冲区，如果字节流太长时，TCP会将其拆分进行发送。当字节流太短时，TCP会等待缓冲区中的字节流达到一定程度时再构成报文发送出去；应用层交给UDP多长的报文，UDP照样发送，既不拆分，也不合并，而是保留这些报文的边界，即一次发送一个报文。

**3．TCP可靠（重传机制），UDP不可靠；**

重传机制：超时重传、快速重传

流量控制：TCP利用滑动窗口机制控制发送方的发送速率

拥塞控制：当出现网络抖动时，TCP会自觉降低发送速度

**4．TCP有序，UDP无序**

消息在传输过程中可能会乱序，后发送的消息可能会先到达，TCP会对其进行重排序，UDP不会

1）发送机在每次发送数据时，会给每个数据包分配一个序列号，并在特定的时间内等待接收机对发送机分配序列号的确认。

2）发送机将已经发送的数据存储在缓存区中，如果特定时间内没有收到接收机对发送机分配序列号的确认，则重复发送此数据包，如果在定时器超时之前收到确认，则将数据包占用的缓存释放。

3）接收机收到数据包后按照序列号将数据包按顺序重组，并传给上层使用。

**5. TCP是重量级的，UDP是轻量级的；**

TCP要建立连接、保证可靠性和有序性，就会传输更多的信息，如TCP的包头比较大。TCP头部需要20字节，UDP头部只要8个字节。

应用场合：TCP一般应用在对可靠性要求比较高的场合，例如http,ftp等等。而UDP一般应用在对实时性要求较高场合，例如视频直播，大文件传输等等

* 1. **三次握手、四次挥手**

（1）第一次握手：Client将标志位SYN置为1，随机产生一个值seq=x，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。

（2）第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=x+1，随机产生一个值seq=y，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。

（3）第三次握手：Client收到确认后，检查ack是否为x+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=y+1，并将该数据包发送给Server，Server检查ack是否为y+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了

SYN=I. 
ACK = l. 网 = x + l. ack y 十 | 
us 0 

为什么握手三次不是两次：防止已失效的连接请求又传送到服务器端，产生错误

--为了实现可靠数据传输， TCP 协议的通信双方， 都必须维护一个序列号，以标识发送出去的数据包中， 哪些是已经被对方收到的。 三次握手的过程即是通信双方相互告知序列号起始值， 并确认对方已经收到了序列号起始值的必经步骤

--如果只是两次握手， 至多只有连接发起方的起始序列号能被确认， 另一方选择的序列号则得不到确认

**Seq的作用：**对于客户端：接收方的ack=发送方seq+1，客户端就能通过ack确定，发送的那个数据包被确认接收了

对于服务端：tcp发送的数据包经过拆分，接收端可以通过seq还原原始的数据包

务 器 
ESTAB- 
主 动 关 勿 LISHED 
数 据 传 这 
H 应 用 
0 
ACK= l. 
= v, ack= u + 1 
数 撫 传 送 
CLOSE 
2 
WAIT 
FIN = 1 ， ACK = l. seq = w. k u + 1 
*ft2MSL 
被 动 夫 
(9 
ACK 

B可能还在发送数据，没有想要关闭数据口的意思，所以FIN与ACK不是同时发送的，而是等到B数据发送完了，才会发送FIN给主机A。

TIME\_WAIT：2MLS MSL：最长报文段寿命，RFC793建议2分钟

1.保证第四次握手的确认能被服务端接收，如果没收到服务端会重传第三次握手

2. A(1.2.3.4:8888)------B(6.7.8.9:9999), 是一个tcp四元组。 当tcp连接关闭后， 四元组释放。 后面的新连接可能会重用到这个四元组 ，新四元组和旧四元组完全一致， 他们的网络包会混乱吗？ 新的连接可能接收到属于原来的那个连接的TCP报文段

因为TCP报文段的最大生存时间是MSL，所以坚持2MSL时间的TIME\_WAIT状态能够确保网络上两个传输方向上尚未被接收到的、迟到的TCP报文段都已经消失（被中转路由器丢弃）。

**Tcp超时重传：**在发送一个数据之后，就开启一个定时器，若是在这个时间内没有收到发送数据的ACK确认报文，则对该报文进行重传

-当一个报文段丢失时，会等待一定的超时周期然后才重传分组，增加了端到端的时延。

-当一个报文段丢失时，在其等待超时的过程中，可能会出现这种情况：其后的报文段已经被接收端接收但却迟迟得不到确认，发送端会认为也丢失了，从而引起不必要的重传，既浪费资源也浪费时间。

**Tcp快速重传：**

采用累计确认机制，即当接收端收到比期望序号大的报文段时，便会重复发送最近一次确认的报文段的确认信号，我们称之为冗余ACK（duplicate ACK）

报文段1成功接收并被确认ACK2，接收端的期待序号为2；当报文段2丢失，报文段3失序到来，与接收端的期望不匹配，接收端重复发送冗余ACK2，报文4、5同上，当连续收到三次冗余ACK，发送端便重传报文2。

Seq # 
1 
2 
3 
4 
5 
2 
ACK 2 
ACK 2 
ACK 2 
ACK 2 
ACK 6 

**TCP、UDP的数据包大小：**

* 1. 在链路层，由以太网的物理特性决定了数据帧的内容最大为1500(不包括帧头和帧尾18)，即MTU为1500；   
     2.在网络层，因为IP包的首部要占用20字节，所以这的MTU为1500－20＝1480；  
     3.在传输层，对于UDP包的首部要占用8字节；TCP包的首部要占用20字节

UDP 包的大小就应该是 1500 - IP头(20) - UDP头(8) = 1472(Bytes)

TCP 包的大小就应该是 1500 - IP头(20) - TCP头(20) = 1460 (Bytes)

当我们的UDP包中的数据多于MTU(1472)时，发送方的IP层需要分片fragmentation进行传输，而在**接收方IP层则需要进行数据报重组**，由于UDP是不可靠的传输协议，如果分片丢失导致重组失败，将导致UDP数据包被丢弃。对于TCP发生组包错误时，该包会被重传，保证可靠传输。

从上面的分析来看，在普通的局域网环境下，UDP的数据最大为1472字节最好(避免分片重组)。

但在网络编程中，Internet中的路由器可能有设置成不同的值(小于默认值)，**Internet上的标准MTU值为576**，所以Internet的UDP编程时数据长度最好在576－20－8＝548字节以内。

**http和https的区别**

http信息是明文传输，https 具有安全性的ssl加密传输协议。

https协议需要到CA申请证书（对服务器公钥的加密）

HTTPS使用端口443，HTTP使用端口80来和TCP/IP进行通信。

http的连接是无状态的（通信双方都不长久的维持对方的任何信息）；SSL/TSL是有状态的， Web服务器和客户端（浏览器）缓存包含加密密钥的会话来认证客户端和服务器。

<https://blog.csdn.net/wangtaomtk/article/details/80917081>

ssl加密发生在应用层和传输层之间

**https工作流程：**

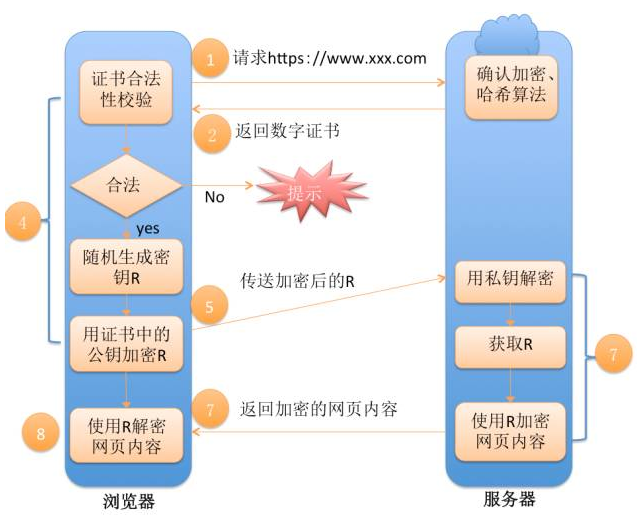
1.浏览器发起往服务器的443端口发起请求，请求携带了浏览器支持的加密算法和哈希算法。

2.服务器将数字证书返回给浏览器，数字证书包含服务器的公钥和证书过期时间，网站网址等内容，客户端通过内置证书列表查询对应机构颁发的公钥，用机构提供的公钥解密证书内容获取服务器公钥、网址、证书过期时间等内容，并验证它们的有效性

3.浏览器生成随机秘钥，并使用服务器的公钥对随机秘钥进行加密，并将随机秘钥传给服务器

4.服务器用自己的私钥解密得到浏览器的随机秘钥，并通过随机秘钥使用对称加密算法加密网页内容发送给浏览器

5.浏览器以R为密钥使用之前约定好的解密算法获取网页内容

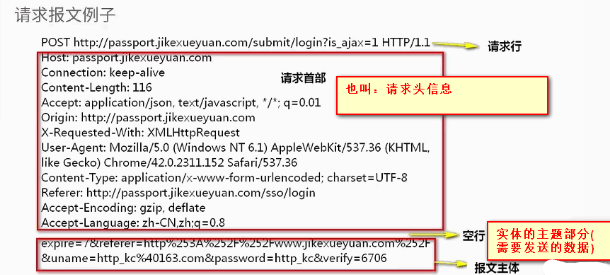


前5步其实就是HTTPS的握手过程，这个过程主要是认证服务端证书（内置的公钥）的合法性。因为非对称加密计算量较大，整个通信过程只会用到一次非对称加密算法（主要是用来保护传输客户端生成的用于对称加密的随机数私钥）。后续内容的加解密都是通过一开始约定好的对称加密算法进行的。

非对称加密算法（RSA）是内容加密的一类算法，它有两个秘钥：公钥与私钥。公钥是公开的钥匙，所有人都可以知道，私钥是保密的，只有持有者知道。通过公钥加密的内容，只能通过私钥解开。非对称加密算法的安全性很高，但是因为计算量庞大，比较消耗性能。

**http请求报文**

格式： 请求行、请求头、空行、请求数据



请求行:请求方式：POST、GET

URL：/DemoEE/form.html

协议版本：HTTP/1.1

请求头：请求头是客户端发送给服务器端的一些信息，使用键值对表示key：value

请求体（get/post）

**响应报文**

格式：响应行、响应头、空行、响应正文

Date: Wed 24 Jun 201S GMT 
Content -Type: textnttml; charset=UTF -8 
close 
X-powered-By; PHP/Sß.6 
Expires: Thu 19 Nov 1981 GMT 
Cache-Control: no-store no-cache must-revalidate 
pre- 
prag ma no-ca 
Set-cookie: expires-Wed Oi-Ju1-20iS GMT; 
Max-Age=6048n• path=/; domain=jikexueyuancom 
Set-cookie: uid=32S7722; expires-Wed Oi-Ju1-2015 GMT; 
Max-Age=604800; path=/; domain=jikexueyuan.com 
{ st : msg 

响应行： Http协议

状态码：

常用的状态码如下：

200 ：请求成功。

302 ：请求重定向。

304 ：请求资源没有改变，访问本地缓存。

500 ：服务器内部错误。通常程序抛异常。

状态信息：状态信息是根据状态码变化而变化的

响应头：响应也都是键值对形式，服务器端将信息以键值对的形式返回给客户端

响应正文：响应体是服务器回写给客户端的页面正文，浏览器将正文加载到内存，然后解析渲染显示页面内容

http1.0/1.1

**缓存处理**，在HTTP1.0中主要使用header里的If-Modified-Since,Expires来做为缓存判断的标准，HTTP1.1则引入了更多的缓存控制策略例如Entity tag，If-Unmodified-Since, If-Match, If-None-Match等更多可供选择的缓存头来控制缓存策略。

**带宽优化及网络连接的使用**，HTTP1.0中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1则在请求头引入了range头域，它允许只请求资源的某个部分，即返回码是206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。

错误通知的管理，在HTTP1.1中新增了24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

**Host头域**，在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。

长连接，HTTP1.0中，一个HTTP请求和响应需要消耗2个RTT(Round-Trip Time，即数据发送时刻到接收到确认的时间差)：客户端会在TCP第三次握手时将HTTP请求报文作为ACK响应给服务器，服务器再以HTTP响应报文作为请求。这样每请求一个文档就要有2倍的RTT的开销，且client和server每次建立连接都要为TCP连接分配缓存和变量，非常消耗资源。所以HTTP1.1采用keep-alive解决了这个问题，一个TCP连接上可以传输多个HTTP请求和响应报文。(HTTP1.1持续连接有两种方式：非流水线、流水线pipeling方式，即client在收到上一次发送的HTTP的响应之前就立即发送下一次请求报文)

HTTP2.0和HTTP1.X相比的新特性

**1.新的二进制格式**（Binary Format），HTTP1.x的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认0和1的组合。基于这种考虑HTTP2.0的协议解析决定采用二进制格式，实现方便且健壮。

**2.多路复用**（MultiPlexing），即连接共享，即每一个request都是是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面。

**3.header压缩**，HTTP1.x的header带有大量信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，差量更新HTTP头部，只发送改变的，既避免了重复header的传输，又减小了需要传输的大小。

**4.服务端推送**（server push），同SPDY一样，HTTP2.0也具有server push功能。

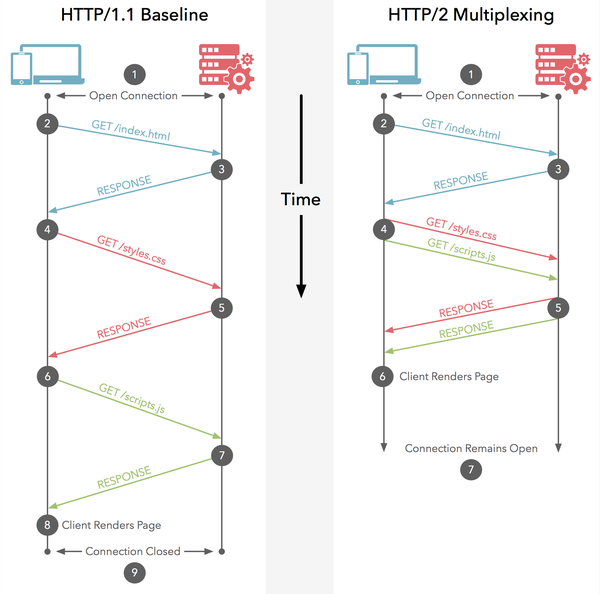
HTTP2.0的多路复用和HTTP1.X中的长连接复用有什么区别？

HTTP/1.\* 一次请求-响应，建立一个连接，用完关闭；每一个请求都要建立一个连接；

HTTP/1.1 Pipeling解决方式为，若干个请求串行化单线程处理，后面的请求等待前面请求的返回才能获得执行机会，一旦有某请求超时等，后续请求只能被阻塞，毫无办法，也就是人们常说的线头阻塞；

HTTP/2 多个请求可同时在一个连接上并行执行。某个请求任务耗时严重，不会影响到其它连接的正常执行；

具体如图：



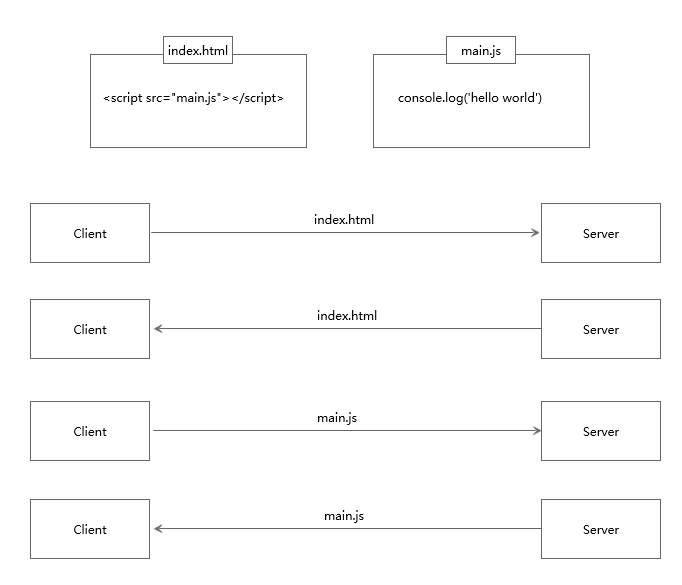
HTTP2.0多路复用有多好？

HTTP 性能优化的关键并不在于高带宽，而是低延迟。TCP 连接会随着时间进行自我「调谐」，起初会限制连接的最大速度，如果数据成功传输，会随着时间的推移提高传输的速度。这种调谐则被称为 TCP 慢启动。由于这种原因，让原本就具有突发性和短时性的 HTTP 连接变的十分低效。

HTTP/2 通过让所有数据流共用同一个连接，可以更有效地使用 TCP 连接，让高带宽也能真正的服务于 HTTP 的性能提升。

服务器推送到底是什么？

服务端推送能把客户端所需要的资源伴随着index.html一起发送到客户端，省去了客户端重复请求的步骤。正因为没有发起请求，建立连接等操作，所以静态资源通过服务端推送的方式可以极大地提升速度。



为什么需要头部压缩？

假定一个页面有100个资源需要加载（这个数量对于今天的Web而言还是挺保守的）, 而每一次请求都有1kb的消息头（这同样也并不少见，因为Cookie和引用等东西的存在）, 则至少需要多消耗100kb来获取这些消息头。HTTP2.0可以维护一个字典，差量更新HTTP头部，大大降低因头部传输产生的流量

**GET/POST**

1.GET提交，请求的数据会附在URL之后（就是把数据放置在HTTP协议头＜request-line＞中），以?分割URL和传输数据，多个参数用&连接;例如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0 %E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密，得出如： %E4%BD%A0%E5%A5%BD，其中％XX中的XX为该符号以16进制表示的ASCII。

POST提交：把提交的数据放置在是HTTP包的包体＜request-body＞中。上文示例中红色字体标明的就是实际的传输数据

因此，GET提交的数据会在地址栏中显示出来，而POST提交，地址栏不会改变

2.传输数据的大小：

首先声明,HTTP协议没有对传输的数据大小进行限制，HTTP协议规范也没有对URL长度进行限制。 而在实际开发中存在的限制主要有：

GET:特定浏览器和服务器对URL长度有限制，例如IE对URL长度的限制是2083字节(2K+35)。对于其他浏览器，如Netscape、FireFox等，理论上没有长度限制，其限制取决于操作系统的支持。

因此对于GET提交时，传输数据就会受到URL长度的限制。

POST:由于不是通过URL传值，理论上数据不受限。但实际各个WEB服务器会规定对post提交数据大小进行限制，Apache、IIS6都有各自的配置。

3.安全性：

POST的安全性要比GET的安全性高。通过GET提交数据，用户名和密码将明文出现在URL上，因为(1)登录页面有可能被浏览器缓存， (2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了。

**Cookie/session**

http请求是无状态的，cookie和session的出现就是为了解决这个问题

cookie保存在客户端客户端上的，比如我们登录网站勾选保存用户名和密码的时候，一般保存的都是cookie，再次登录的时候浏览器直接将对应的cookie发送到服务端验证，注意的是密码不存在cookie里，cookie里只存用户登录账号和自动登录标志位。

session保存在服务器端内存中， 存储和用户会话相关的数据（身份信息、登录状态、权限列表、购物车数据等），并通过Session ID进行认证，当客户端第一次请求session对象时候，服务器会为客户端创建一个session，并生成对应sessionID，用来标识该session对象，并把Session ID返回给客户端保存在客户端cookie中。客户端再访问时通过验证从客户端发送过来的Session ID进行身份认证，获取相应的session数据，返还给请求者（Servlet）使用。

当我们把浏览器的cookie禁止后，web服务器会采用URL重写的方式传递Sessionid，我们就可以在地址栏看到 sessionid=KWJHUG6JJM65HS2K6之类的字符串。

同一客户端机器多次请求同一个资源，session一样吗？

一般来说，每次请求都会新创建一个session。

Session删除的时间是：

1）Session超时：超时指的是连续一定时间服务器没有收到该Session所对应客户端的请求，并且这个时间超过了服务器设置的Session超时的最大时间。

2）程序调用HttpSession.invalidate()

3）服务器关闭或服务停止



session 
UserBean ub 
ubb. getUserBean(u) ; 
request. getSession() . setAttribute( "userlnfo", ub); 
MyCartBO mcb = .getAttribute("mycart"); 

<https://www.zhihu.com/question/19786827>

**http状态码**

1xx：指示信息–表示请求已接收，继续处理。

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。

301 永久性重定向

该状态码表示请求的资源已被分配到新的URI，这就代表你之前保存的书签已经没用了，需要换成新的URI。

302 临时性重定向

该状态码表示请求的资源已被分配了新的 URI， 希望用户（本次） 能使用新的 URI 访问，也就是说你不需要去更新保存的书签，只是临时性质。

303 状态码

该状态码表示由于请求对应的资源存在着另一个 URI， 会将POST重定向为GET方法定向获取请求的资源

307

当客户端的POST请求收到服务端307状态码响应时，需要跟用户询问是否应该在新URI上发起POST方法，307是不会把POST转为GET的。

304 文件未变化

如果客户端发送了一个带条件的GET 请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个304状态码。简单的表达就是：服务端已经执行了GET，但文件未变化。304 状态码返回时， 不包含任何响应的主体部分。

4xx：客户端错误–请求有语法错误或请求无法实现。

400 Bad Request：客户端请求有语法错误，不能被服务器所理解。

401 Unauthorized：请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用。

403 Forbidden：服务器收到请求，但是拒绝提供服务。

404 Not Found：请求资源不存在，举个例子：输入了错误的URL。

5xx：服务器端错误–服务器未能实现合法的请求。

500 Internal Server Error：服务器发生不可预期的错误。可能用户权限的问题导致，或者是数据库连接出现了错误

502 Bad Gateway ：网关错误，通俗点说就是和web服务器通信失败，开启了防火墙，web服务器请求太多，响应不了这个请求，

503 Server Unavailable：服务器当前不能处理客户端的请求，一段时间后可能恢复正常

504错误代表网关超时 (Gateway timeout)，是指服务器作为网关或代理，但是没有及时从上游服务器收到请求，一般指nginx做反向代理服务器时，所连接的服务器tomcat无响应导致的。。