腾讯云负载均衡

延伸阅读

产品文档

【版权声明】

©2013-2017 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

文档声明 2
延伸阅读 4
HTTP长连接说明 4
WebSocket原理说明 8
SSL 原理说明 13
会话保持原理 20
cookie原理说明 25
HTTP返回值说明 30
SSL证书链说明 32
SSL单向认证和双向认证说明 34

延伸阅读

HTTP长连接说明

目前腾讯云负载均衡对七层负载均衡的HTTP长连接配置，可设置为默认值75s（该配置项后续会开放），用户可对不同的负载均衡实例进行自定义配置。那么，HTTP长连接、短连接究竟是什么？

1. HTTP协议与TCP/IP协议的关系

HTTP的长连接和短连接本质上是TCP长连接和短连接。HTTP属于应用层协议，在传输层使用TCP协议，在网络层使用IP协议。 IP协议主要解决网络路由和寻址问题，TCP协议主要解决如何在IP层之上可靠地传递数据包，使得网络上接收端收到发送端所发出的所有包，并且顺序与发送顺序一致。TCP协议是可靠的、面向连接的。

2. 如何理解HTTP协议是无状态的

HTTP协议是无状态的，指的是协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态。也就是说，打开一个服务器上的网页和上一次打开这个服务器上的网页之间没有任何联系。HTTP是一个无状态的面向连接的协议，无状态不代表HTTP不能保持TCP连接，更不能代表HTTP使用的是UDP协议（无连接）。

3. 什么是长连接、短连接？

在HTTP/1.0中默认使用短连接。也就是说，客户端和服务器每进行一次HTTP操作，就建立一次连接，任务结束就中断连接。当客户端浏览器访问的某个HTML或其他类型的Web页中包含有其他的Web资源（如JavaScript文件、图像文件、CSS文件等），每遇到这样一个Web资源，浏览器就会重新建立一个HTTP会话。

而从HTTP/1.1起，默认使用长连接，用以保持连接特性。使用长连接的HTTP协议，会在响应头加入这行代码：

```
Connection:keep-alive
```

在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，客户端再次访问这个服务器时，会继续使用这一条已经建立的连接。Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。实现长连接需要客户端和服务端都支持

版权所有：腾讯云计算（北京）有限责任公司

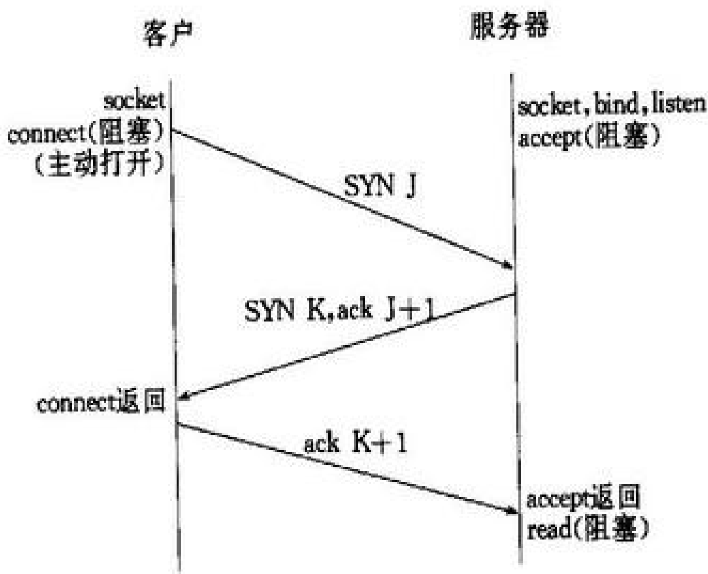
长连接。

HTTP协议的长连接和短连接，实质上是TCP协议的长连接和短连接。

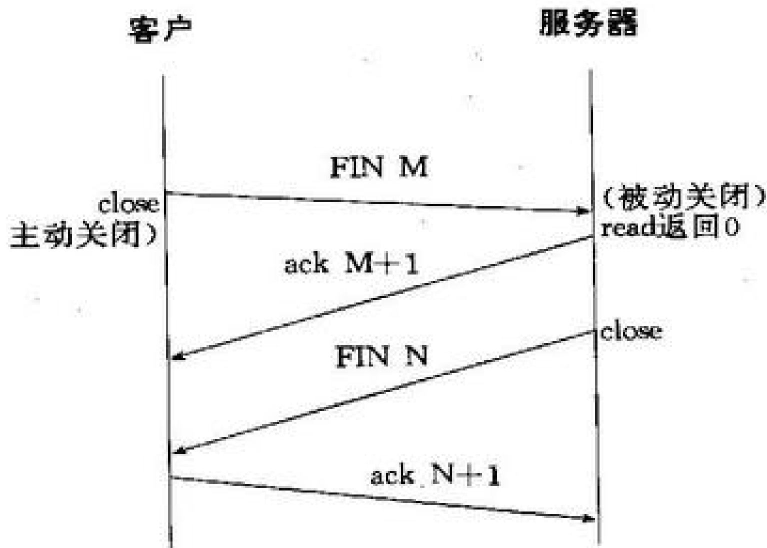
3.1. TCP连接

当网络通信时采用TCP协议时，在真正的读写操作之前，客户端与服务器端之间必须建立一个连接，当读写操作完成后，双方不再需要这个连接时可以释放这个连接。连接的建立依靠“三次握手”，而释放则需要“四次握手”，所以每个连接的建立都是需要资源消耗和时间消耗的。

经典的三次握手建立连接示意图：



经典的四次握手关闭连接示意图：



3.2. TCP短连接

模拟一下TCP短连接的情况：client向server发起连接请求，server接到请求，然后双方建立连接。client向server发送消息，server回应client，然后一次请求就完成了。这时候双方任意都可以发起close操作，不过一般都是client先发起close操作。上述可知，短连接一般只会在 client/server间传递一次请求操作。

短连接的优点是：管理起来比较简单，存在的连接都是有用的连接，不需要额外的控制手段。

3.3. TCP长连接

我们再模拟一下长连接的情况：client向server发起连接，server接受client连接，双方建立连接，client与server完成一次请求后，它们之间的连接并不会主动关闭，后续的读写操作会继续使用这个连接。

TCP的保活功能主要为服务器应用提供。如果客户端已经消失而连接未断开，则会使得服务器上保留一个半开放的连接，而服务器又在等待来自客户端的数据，此时服务器将永远等待客户端的数据。保活功能就是试图在服务端器端检测到这种半开放的连接。

如果一个给定的连接在两小时内没有任何动作，服务器就向客户发送一个探测报文段，根据客户端主机响应探测4个客户端状态：

- 客户主机依然正常运行，且服务器可达。此时客户的TCP响应正常，服务器将保活定时器复位。
- 客户主机已经崩溃，并且关闭或者正在重新启动。上述情况下客户端都不能响应TCP。服务端将无法收

到客户端对探测的响应。服务器总共发送10个这样的探测，每个间隔75秒。若服务器没有收到任何一个响应，它就认为客户端已经关闭并终止连接。

- 客户端崩溃并已经重新启动。服务器将收到一个对其保活探测的响应，这个响应是一个复位，使得服务器终止这个连接。
- 客户机正常运行，但是服务器不可达。这种情况与第二种状态类似。

4. 长连接和短连接的优点和缺点

由上可以看出，长连接可以省去较多的TCP建立和关闭的操作，减少浪费，节约时间。对于频繁请求资源的客户端适合使用长连接。在长连接的应用场景下，client端一般不会主动关闭连接，当client与server之间的连接一直不关闭，随着客户端连接越来越多，server会保持过多连接。这时候server端需要采取一些策略，如关闭一些长时间没有请求发生的连接，这样可以避免一些恶意连接导致server端服务受损；如果条件允许则可以限制每个客户端的最大长连接数，这样可以完全避免恶意的客户端拖垮整体后端服务。

短连接对于服务器来说管理较为简单，存在的连接都是有用的连接，不需要额外的控制手段。但如果客户请求频繁，将在TCP的建立和关闭操作上浪费较多时间和带宽。

长连接和短连接的产生在于client和server采取的关闭策略。不同的应用场景适合采用不同的策略。

WebSocket原理说明

众所周知，Web应用的通信过程通常是客户端通过浏览器发出一个请求，服务器端接收请求后进行处理并返回结果给客户端，客户端浏览器将信息呈现。这种机制对于信息变化不是特别频繁的应用可以良好支撑，但对于实时要求高、海量并发的应用来说显得捉襟见肘，尤其在当前业界移动互联网蓬勃发展的趋势下，高并发与用户实时响应是Web应用经常面临的问题，比如金融证券的实时信息、Web导航应用中的地理位置获取、社交网络的实时消息推送等。

传统的请求-响应模式的Web开发在处理此类业务场景时，通常采用实时通讯方案。比如常见的轮询方案，其原理简单易懂，就是客户端以一定的时间间隔频繁请求的方式向服务器发送请求，来保持客户端和服务器端的数据同步。其问题也很明显：当客户端以固定频率向服务器端发送请求时，服务器端的数据可能并没有更新，带来很多无谓请求，浪费带宽，效率低下。

基于Flash，AdobeFlash通过自己的Socket实现完成数据交换，再利用Flash暴露出相应的接口给JavaScript调用，从而达到实时传输目的。此方式比轮询要高效，且因为Flash安装率高，应用场景广泛。然而，移动互联网终端上Flash的支持并不好：IOS系统中无法支持Flash，Android虽然支持Flash但实际的使用效果差强人意，且对移动设备的硬件配置要求较高。2012年Adobe官方宣布不再支持Android4.1+系统，宣告了Flash在移动终端上的死亡。

传统的Web模式在处理高并发及实时性需求的时候，会遇到难以逾越的瓶颈，需要一种高效节能的双向通信机制来保证数据的实时传输。在此背景下，基于HTML5规范的、有Web TCP之称的 WebSocket应运而生。早期HTML5并没有形成业界统一的规范，各个浏览器和应用服务器厂商有着各异的类似实现，如IBM的MQTT、Comet开源框架等。直到2014年，HTML5终于尘埃落地，正式落实为实际标准规范，各个应用服务器及浏览器厂商逐步开始统一，在
Ja
vaE
E7中也
实现了WebSo
cket协议。至此无论是客户端
还是服务端的WebSocket都已完备。用户可以查阅HTML5规范
，熟悉新的HTML协议规范及WebSocket支持。

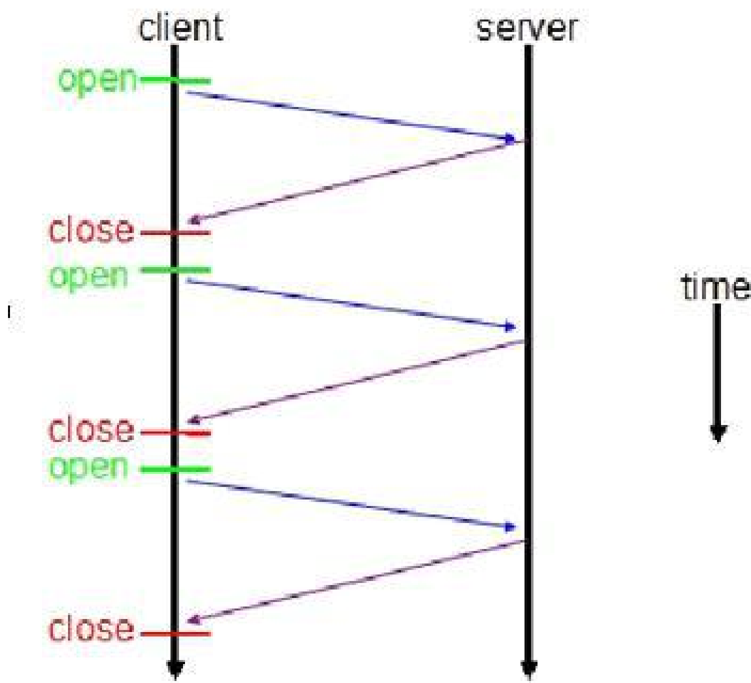
WebSocket 机制

以下简要介绍一下WebSocket的原理及运行机制。

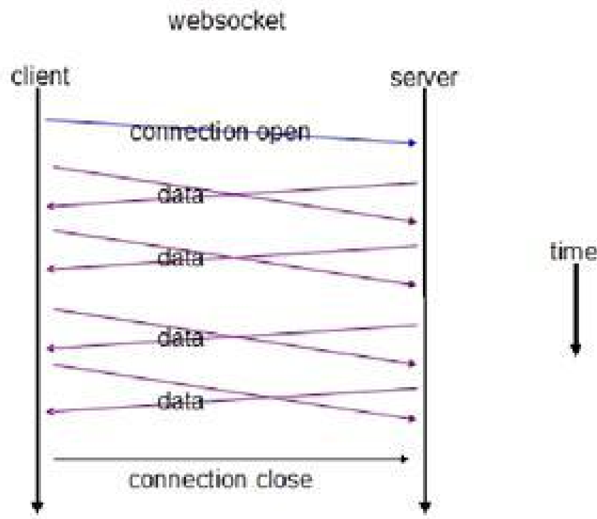
WebSocket是HTML5下一种新的协议。它实现了浏览器与服务器全双工通信，能更好的节省服务器资源和带宽并达到实时通讯的目的。它与HTTP一样通过已建立的TCP连接来传输数据，但是它和HTTP最大不同是：

- WebSocket是一种双向通信协议。在建立连接后，WebSocket服务器端和客户端都能主动向对方发送或接收数据，就像Socket一样；
- WebSocket需要像TCP一样，先建立连接，连接成功后才能相互通信。

传统HTTP客户端与服务器请求响应模式如下图所示：



WebSocket模式客户端与服务器请求响应模式如下图：



上图对比可以看出，相对于传统HTTP每次请求-应答都需要客户端与服务端建立连接的模式，WebSocket是类似Socket的TCP长连接通讯模式。一旦WebSocket连接建立后，后续数据都以帧序列的形式传输。在客户端断开WebSocket连接或Server端中断连接前，不需要客户端和服务端重新发起连接请求。在海量并发及客户端与服务器交互负载流量大的情况下，极大的节省了网络带宽资源的消耗，有明显的性能优势，且客户端发送和接受消息是在同一个持久连接上发起，实时性优势明显。

相比HTTP长连接，WebSocket有以下特点：

- 是真正的全双工方式，建立连接后客户端与服务器端是完全平等的，可以互相主动请求。而HTTP长连接基于HTTP，是传统的客户端对服务器发起请求的模式。
- HTTP长连接中，每次数据交换除了真正的数据部分外，服务器和客户端还要大量交换HTTP header，信息交换效率很低。Websocket协议通过第一个request建立了TCP连接之后，之后交换的数据都不需要发送 HTTP header就能交换数据，这显然和原有的HTTP协议有区别所以它需要对服务器和客户端都进行升级才能实现（主流浏览器都已支持HTML5）。此外还有 multiplexing、不同的URL可以复用同一个WebSocket连接等功能。这些都是HTTP长连接不能做到的。

下面再通过客户端和服务端交互的报文对比WebSocket通讯与传统HTTP的不同点：

在客户端，new WebSocket实例化一个新的WebSocket客户端对象，请求类似 ws://yourdomain:port/path 的服务端WebSocket URL，客户端WebSocket对象会自动解析并识别为WebSocket请求，并连接服务端端口，执行双方握手过程，客户端发送数据格式类似：

GET/webfin/websocket/ HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
Origin: http://localhost:8080
Sec-WebSocket-Version: 13

可以看到，客户端发起的WebSocket连接报文类似传统HTTP报文，

Upgrade：websocket

参数值表明这是WebSocket类型请求，

Sec-WebSocket-Key

是WebSocket客户端发送的一个 base64编码的密文，要求服务端必须返回一个对应加密的

Sec-WebSocket-Accept

应答，否则客户端会抛出

Error during WebSocket handshake

错误，并关闭连接。

服务端收到报文后返回的数据格式类似：

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=

Sec-WebSocket-Accept

的值是服务端采用与客户端一致的密钥计算出来后返回客户端的，

HTTP/1.1 101 Switching Protocols

表示服务端接受WebSocket协议的客户端连接，经过这样的请求-
响应处理后，两端的WebSocket连接握手成功,
后续就可以进行TCP通讯了。用户可以查阅[WebSocket协议栈](#)
了解WebSocket客户端和服务端更详细的交互数据格式。

在开发方面，WebSocket API 也十分简单：只需要实例化
WebSocket，创建连接，然后服务端和客户端就可以相互发送和响应消息。在WebSocket
实现及案例分析部分可以看到详细的 WebSocket API 及代码实现。

腾讯云公网有日租类型七层负载均衡转发部分支持Websocket，目前包括英魂之刃、银汉游戏等多家企业已接入使用。当出现不兼容问题时，请修改websocket配置，websocket server不校验下图中圈出的字段：

```
GET / HTTP/1.0
Upgrade: websocket
Connection: upgrade
Host: 21319
Pragma: no-cache
Cache-Control: no-cache
Origin: chrome-extension://pfdhoblngboilpfeibdedpjgfnlcodoo
Sec-WebSocket-Version: 13
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Sec-WebSocket-Key: qhiuFIg2USbwzjyKs28zrg==
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: TgxnYhlXJHsvWZbPJEleQA8+XNo=
```

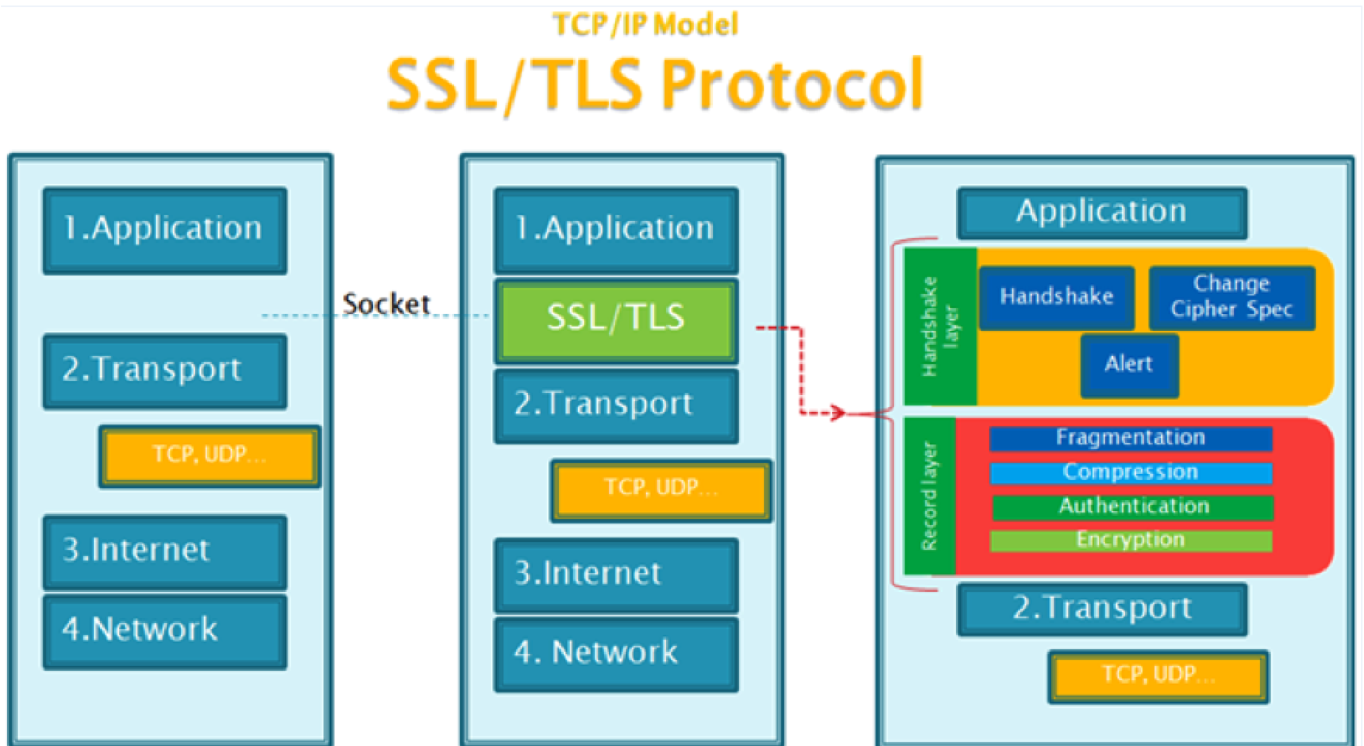
一个使用WebSocket应用于视频的业务思路如下：

- 使用心跳维护websocket链路，探测客户端端的网红/主播是否在线
- 设置负载均衡7层的proxy_read_timeout默认为60s
- 设置心跳为50s，即可长期保持Websocket不断开

近期Websocket将开放自定义配置，敬请期待。

SSL 原理说明

SSL/TLS是一个介于应用层（HTTP 协议）与传输层（TCP
协议）之间的一个可选协议，其协议架构可参照下图：



当HTTP通信不使用 SSL/TLS 时，所有信息均以明文形式传播，会有以下风险：

- 窃听风险（eavesdropping）：第三方可以获得通信内容
- 篡改风险（tampering）：第三方可以修改通信内容
- 冒充风险（pretending）：第三方可以冒充他人身份参与通信

为了解决这些通信风险，SSL/TLS 协议应运而生。协议设计的目标为：

- 所有信息都是加密传播，第三方无法窃听。
- 具有校验机制，一旦被篡改，通信双方可以立即发现。
- 配备身份证书，防止身份被冒充。

目前，主流浏览器都已经支持了 SSL/TLS 的支持。

1. SSL/TLS 协议基本运行过程

版权所有：腾讯云计算（北京）有限责任公司

SSL/TLS 协议的基本思路是采用公钥加密的方法。即客户端先向服务器端索要公钥，然后使用公钥加密信息并发送至服务器端；服务器收到密文后，用自己的私钥解密。

这里需要解决两个问题：

- 如何保证公钥不被篡改？
  解决方法：将公钥放在数字证书中。只要证书是可信的，公钥就是可信的。
- 公钥加密计算量太大，如何减少耗用的时间？
  解决方法：每一次对话（session），客户端和服务器端都生成一个"对话密钥"（session key），用它来加密信息。由于"对话密钥"是对称加密，所以运算速度非常快，而服务器公钥只用于加密"对话密钥"本身，这样就减少了加密运算的消耗时间。

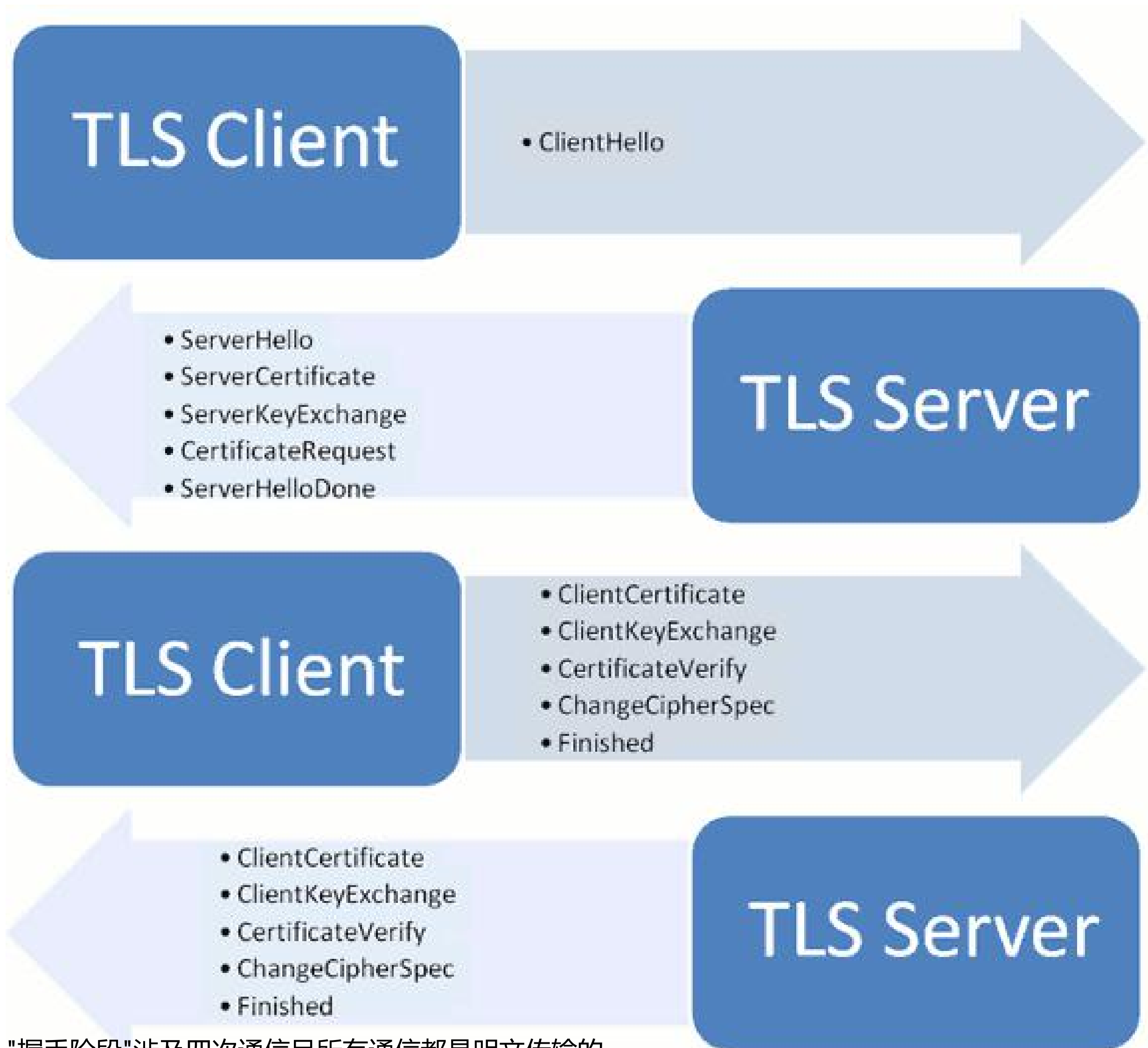
SSL/TLS协议的基本过程是这样的：

- 客户端向服务器端索要并验证公钥
- 双方协商生成"对话密钥"
- 双方采用"对话密钥"进行加密通信。

上面过程的前两步，又称为"握手阶段"（handshake）。

2. 握手阶段的详细过程

版权所有：腾讯云计算（北京）有限责任公司



"握手阶段"涉及四次通信且所有通信都是明文传输的。

2.1. 客户端发出请求（ClientHello）

客户端（通常是浏览器）先向服务器发出加密通信的请求，通常被称为ClientHello请求。

在这一步，客户端主要向服务器提供以下信息。

- 支持的协议版本，比如TLS 1.0
- 一个客户端生成的随机数，稍后用于生成"对话密钥"
- 支持的加密方法，比如RSA公钥加密
- 支持的压缩方法

这里需要注意的是，客户端发送的信息之中不包括服务器的域名。也就是说，理论上服务器只能包含一个网站，否则会分不清应该向客户端提供哪一个网站的数字证书。这就是为什么通常一台服务器只能有一张数字证书

的。

对于虚拟主机的用户来说，这当然很不方便。2006年，TLS协议加入了一个 Server Name Indication 扩展，允许客户端向服务器提供它所请求的域名。

2.2. 服务器回应（SeverHello）

服务器收到客户端请求后，向客户端发出回应，通常被称为SeverHello。服务器的回应包含以下内容：

- 确认使用的加密通信协议版本，比如TLS 1.0版本。如果浏览器与服务器支持的版本不一致，服务器将关闭加密通信
- 一个服务器生成的随机数，稍后用于生成"对话密钥"
- 确认使用的加密方法，比如RSA公钥加密
- 服务器证书

除了以上信息，若服务器需要确认客户端的身份，则会再包含一项请求，要求客户端提供"客户端证书"。比如，金融机构往往只允许认证客户连入自己的网络，就会向正式客户提供USB密钥，里面就包含了一张客户端证书。

2.3. 客户端回应

客户端收到服务器回应以后会首先验证服务器证书。如果证书不是由可信机构颁布、证书中的域名与实际域名不一致或者证书已经过期，就会向访问者显示一个警告，由其选择是否还要继续通信。
如果证书没有问题，客户端就会从证书中取出服务器的公钥并向服务器发送以下信息：

- 一个随机数。该随机数用服务器公钥加密，防止被窃听。
- 编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。
- 客户端握手结束通知，表示客户端的握手阶段已经结束。这一项同时也是前面发送的所有内容的hash值，用来供服务器校验。

上面第一项的随机数，是整个握手阶段出现的第三个随机数，又称"pre-master key"。有了它以后，客户端和服务器就同时有了三个随机数，接着双方就用事先商定的加密方法，各自生成本次会话所用的同一把"会话密钥"。

版权所有：腾讯云计算（北京）有限责任公司

对于RSA密钥交换算法来说，pre-master-key本身就是一个随机数，再加上hello消息中的随机，三个随机数通过一个密钥导出器最终导出一个对称密钥。

pre-master的存在原因是SSL协议不信任每个主机都能产生完全随机的随机数。客户端和服务器加上pre-master三个随机数一同生成的密钥不容易被猜出了。因为一个伪随机可能完全不随机，可是三个伪随机则十分接近随机。

此外，如果前一步，服务器要求客户端证书，客户端会在这一步发送证书及相关信息。

2.4. 服务器的最后回应

服务器收到客户端的第三个随机数pre-master-key之后，计算生成本次会话所用的"会话密钥"。然后，向客户端最后发送如下信息：

- 编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。
- 服务器握手结束通知，表示服务器的握手阶段已经结束。这一项同时也是前面发送的所有内容的hash值，用来供客户端校验。

至此，整个握手阶段全部结束。接下来，客户端与服务器进入加密通信，完全是使用HTTP协议，只不过用"会话密钥"加密内容。

使用一个简单的例子来说明上面的内容：假设A与B通信，A是SSL客户端，B是SSL服务器端，加密后的消息放在方括号[]里，以突出明文消息的区别。双方处理动作的说明用圆括号（）括起。

- A：

“我想和你安全的通话，我这里的对称加密算法有DES,RC5；密钥交换算法有RSA和DH；摘要算法有MD5和SHA。”

- B：

“我们用DES－RSA－SHA这对组合好了。这是我的证书，里面有我的名字和公钥，你拿去验证一下我的身份。”

把证书发给A。

“目前没有别的可说的了。”

- A：

查看证书上B的名字是否无误，并通过手头早已有的CA的证书验证了B的证书的真实性，如果其中一项有误，发出警告并断开连接，这一步保证了B的公钥的真实性。

产生一份秘密消息，这份秘密消息处理后将用作加密密钥，加密初始化向量（IV）和hmac的密钥。将这份秘密消息（协议中称为 per_master_secret）用B的公钥加密，封装成称作ClientKeyExchange的消息。由于用了B的公钥，保证了第三方无法窃听。

“我生成了一份秘密消息，并用你的公钥加密了，给你。” 把ClientKeyExchange发给B。 “注意，下面我就要用加密的办法给你发消息了！”

将秘密消息进行处理，生成加密密钥，加密初始化向量和hmac的密钥。

[我说完了。]

- B：

用自己的私钥将ClientKeyExchange中的秘密消息解密出来，然后将秘密消息进行处理，生成加密密钥，加密初始化向量和hmac的密钥，这时双方已经安全的协商出一套加密办法了。

“注意，我也要开始用加密的办法给你发消息了！”

[我说完了]

版权所有：腾讯云计算（北京）有限责任公司

- A:

[我的秘密是...]

- B:

[其它人不会听到的...]

会话保持原理

1. 什么是会话保持？

会话保持是负载均衡最常见的问题之一，也是一个相对比较复杂的问题。会话保持有时候又叫做粘滞会话(Sticky Sessions)。会话保持是指在负载均衡器上的一种机制，可以识别客户端与服务器之间交互过程的关连性，在作负载均衡的同时还保证一系列相关连的访问请求会保持分配到一台服务器上。

2. 什么时候需要会话保持？

在讨论这个问题前，我们必须先花点时间弄清楚一些概念：什么是连接（Connection）、什么是会话（Session），以及这二者之间的区别。需要特别强调的是，如果我们仅仅是谈论负载均衡，会话和连接往往具有相同的含义。

从简单的角度来看，如果用户需要登录，那么就可以简单的理解为会话；如果不需要登录，那么就是连接。

对于同一个连接中的数据包，负载均衡会将其进行NAT转换后，转发至后端固定的服务器进行处理。负载均衡系统内部会专门有一张表来记录这些连接的状况，包括：[源IP：端口]、[目的IP：端口]、[服务器IP：端口]、空闲超时时间（Idle Timeout）等等。由于负载均衡内部记录连接状态的这张表需要消耗系统的内存资源，因此这张表不可能无限大，所有传统厂商都会有一定的限制。这张表的大小一般称之为最大并发连接数，也就是系统同时能够容纳的连接数量。负载均衡的当前连接状态表项中，设计了一个空闲超时时间（Idle Timeout）的参数。当该连接在Idle Timeout内无流量通过时，负载均衡会自动删除该连接条目，释放系统资源。

删除连接后，客户端的请求将无法保证继续发往同一个后端服务器，需要遵循负载均衡器的流量分发策略。

在某些要求登录状态的情境下，要求客户端和服务器之间保持一个会话（session）以记录客户端的各种信息。比如在大多数电子商务的应用系统或者需要进行用户身份认证的在线系统中，一个客户与服务器经常经过好几次的交互过程才能完成一笔交易或者是一个请求的完成。由于这几次交互过程是密切相关的，服务器在进行这些交互过程的某一个交互步骤时往往需要了解上一次或上几次的交互过程处理结果，这就要求所有这些相关的交互过程都由一台服务器完成，而不能被负载均衡器分散到不同的服务器上。否则可能出现异常情景：

- 客户端输入了正确的用户名和口令，但却反复跳到登录页面；
- 用户输入了正确的验证码，但是总提示验证码错误；

版权所有：腾讯云计算（北京）有限责任公司

- 客户端放入购物车的物品丢失
- ...

因此会话保持机制的意义就在于，确保在合适的情境下，将来自相同客户端的请求转发至后端相同的服务器进行处理。换句话说，就是将客户端与服务器之间建立的多个连接，都发送到相同的服务器进行处理。如果在客户端和服务器之间部署了负载均衡设备，很有可能这多个连接会被转发至不同的服务器进行处理。如果服务器之间没有会话信息的同步机制，会导致其他服务器无法识别用户身份，造成用户在和应用系统发生交互时出现异常。

负载均衡希望将来自客户端的连接、请求均衡的转发至后端的多台服务器，以避免单台服务器负载过高；而会话保持机制却要求将某些请求转发至同一台服务器进行处理。因此，在实际的部署环境中，我们要根据应用环境的特点，选择适当的会话保持机制。

3. 会话保持的分类

3.1. 简单会话保持（四层会话保持）

简单会话保持（也称作基于源地址的会话保持、基于IP的会话保持）是指负载均衡器在作负载均衡时根据访问请求的源地址作为判断关连会话的依据。对来自同一IP地址的所有访问请求在作负载均时都会被保持到一台服务器上去。

简单会话保持中一个很重要的参数就是连接超时值，负载均衡器会为每一个处于保持状态中的会话设定一个时间值。若一个会话从上一次完成到下次再来之间的间隔时间小于超时值时，负载均衡器将会将新的连接进行会话保持；但如果这个间隔大于该超时值，负载均衡器会将新来的连接认为是新的会话然后进行负载平衡。

简单会话保持实现简单，只需要根据数据包三、四层的信息就可以实现，效率比较高。

```
NginX对简单会话保持的支持
ip_hash
每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session 的问题。
例如:
         upstream bakend {
         ip_hash;
         server 192.168.0.14:88;
         server 192.168.0.15:80;
                                 }
```

但此种方式存在的问题就在于，当多个客户端通过代理或地址转换的方式访问服务器时，由于来源地址一样，请求都被分配到同一台服务器上，会导致服务器之间的负载严重失衡。

另外一种情况是，同一个客户端产生大量并发，要求分配到多个服务器上处理的同时进行会话保持。这时基于客户端源地址的会话保持方法也会导致负载均衡失效。

以上情况出现时，就必须要考虑使用其他的会话保持方式。

3.2. 存会话（session）的会话保持

此种方式通过多个后端服务器共享session的方式，实现与负载均衡同时的会话保持。主要有以下几种形式：

1) 数据库存放

Session信息存储到数据库表以实现不同应用服务器间Session信息的共享。此种方式适合数据库访问量不大的网站。

- 优点：实现简单
- 缺点：由于数据库服务器相对于应用服务器更难扩展且资源更为宝贵，在高并发的Web应用中，最大的性能瓶颈通常出现在数据库服务器。因此如果将Session存储到数据库表，频繁的数据库操作会影响业务。

2) 文件系统存放

通过文件系统（比如NFS）来实现各台服务器间的Session共享。此种方式适合并发量不大的网站。

- 优点：各台服务器只需要mount存储Session的磁盘即可，实现较为简单。
- 缺点：NFS对高并发读写的性能并不高，在硬盘I/O性能和网络带宽上存在较大瓶颈，尤其是对于Session这样的小文件的频繁读写操作。

3) Memcached存放

利用Memcached来保存Session数据，直接通过内存的方式读取。

版权所有：腾讯云计算（北京）有限责任公司

- 优点：效率高，在读写速度上会比存放在文件系统时快很多，而且多个服务器共用Session也更加方便，将这些服务器都配置成使用同一组memcached服务器就可以，减少了额外的工作量。
- 缺点：一旦宕机内存中的数据将会丢失，但对Session数据来说并不是严重的问题。如果网站访问量太大、Session太多的时候memcached会将不常用的部分删除，但是如果用户隔离了一段时间之后继续使用，将会发生读取失败的问题。

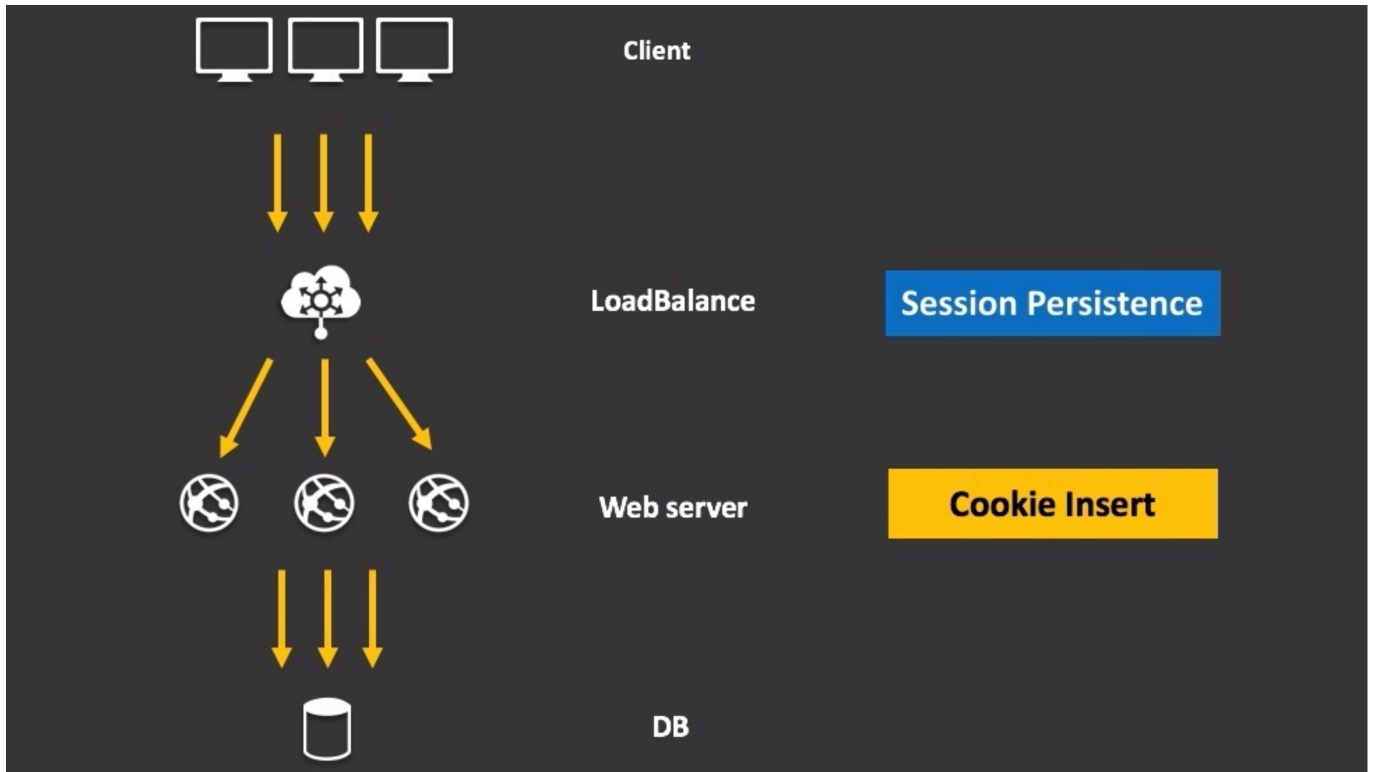
3.3. 基于cookie的会话保持（七层会话保持）

在基于cookie模式下负载均衡器负责插入cookie，后端服务器无需作出任何修改。当客户端进行第一次请求时，客户端的HTTP request（不带cookie）进入负载均衡器，
CLB根据负载平衡算法策略选择后端一台服务器，并将请求发送至该服务器；后端服务器的HTTP
response（不带cookie）被发回给负载均衡器。接下来负载均衡器将向该后端服务器插入cookie并将HTTP
response返回到客户端。

当客户请求再次发生时，客户HTTP
request（带有上次负载均衡器插入的cookie）进入CLB，然后CLB读出cookie里的会话保持数值，将HTTP request（带有与上面同样的cookie）发到指定的服务器，然后后端服务器进行请求回复；由于服务器并不写入cookie，HTTP response将不带cookie，该HTTP
response再次经过进入CLB时，CLB将写入更新后的会话保持cookie。

腾讯云CLB的7层会话保持能力，就是基于这样的cookie插入的方式实现的。

版权所有：腾讯云计算（北京）有限责任公司


Client
LoadBalance
Session Persistence
Web server
Cookie Insert
DB

cookie原理说明

什么是cookie?

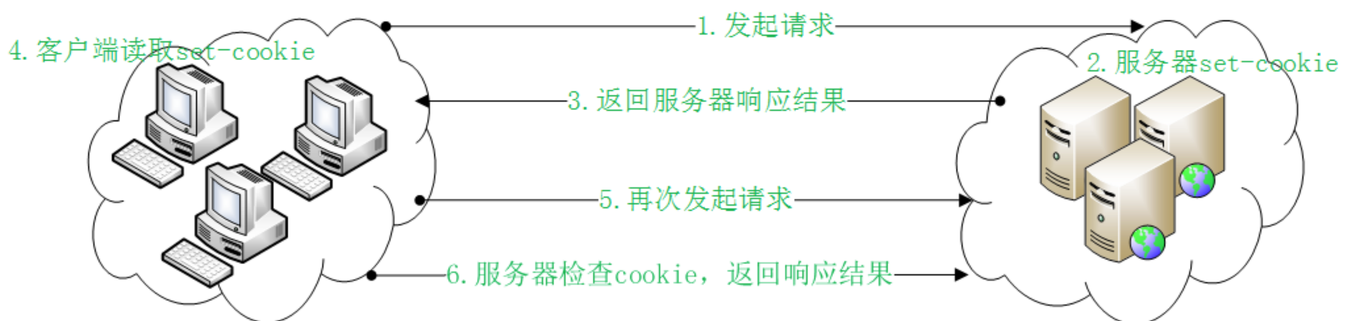
HTTP协议是无状态的，也就是说客户端和服务器端不需要建立持久的连接。由于客户端和服务器端的连接是基于一种请求应答模式，即客户端和服务器建立一个连接-客户端提交一个请求-服务器端收到请求后返回一个响应，然后二者就断开连接。

若客户端和服务器在完成一次请求以后就断开了连接，二者之间就不再有任何关系了；那么，当用户在页面1进行了登录后跳转到了同一个Web应用的页面2时，如何在页面2知道用户已经进行了登录呢？亦即当客户端再次发起请求的时候，服务器端如何判断两次不同的请求来自同一个客户端呢？

HTTP协议下，服务器是无法区分每一次请求之间的联系的。要判断这种联系就需要有一个状态来标识每一次请求，如果两次请求的状态标识是一样的，这就表明这两个请求是从同一个客户端发起的。

Cookie就是这样一个用来标识每一次请求的状态位。经过多年的发展Cookie变得越来越规范，后来直接成为了一个通用标准。

cookie的工作原理



1) 当首次向腾讯云发起请求时，HTTP请求头如下：

Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/;q=0.8
Accept-Encoding:gzip, deflate, sdch
Accept-Language:en,zh-CN;q=0.8,zh;q=0.6
Connection:keep-alive
Host:cloud.tencent.com

2) 请求到达腾讯云的服务器以后，腾讯云的服务器生成响应，并在响应的头部写入cookie信息：

Set-Cookie:BD_HOME=1; path=/
Set-Cookie:__bsi=14934756243064632384_00_0_I_R_174_0303_C02F_N_I_I_0; expires=Thu, 19-Nov-15 14:14:50 GMT; domain=www.qcloud; path=/
Set-Cookie:BDSVRTM=172; path=/

3) 当客户端浏览器接收到响应头以后，会将cookie信息写入本地进行管理。

4) 再次向服务器发起请求时，客户端通过发送一个带有Cookie: name=value; name2=value2的HTTP请求头将之前存在本地的cookie一起发送过去。请求的头部信息为：

Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/;q=0.8
Accept-Encoding:gzip, deflate, sdch
Accept-Language:en,zh-CN;q=0.8,zh;q=0.6
Connection:keep-alive
Cookie:BD_HOME=1; BDSVRTM=0; BD_LAST_QID=1507196234531915875957057
Host:cloud.tencent.com

5) 服务器接收到请求以后，从请求头中获得cookie信息，分析cookie数据后向客户端返回响应。

以上就是cookie在客户端和服务器之间进行传递信息的基本过程。

cookie的生命周期

cookie是有生命周期的。一旦到了cookie的失效日期，客户端的cookie就会被删除，服务器在创建cookie时可以控制一个cookie可以在客户端“存活”多长时间。在以下几种情况下，cookie都会结束它自己的生命周期：

- 未指定过期时间的cookie。当服务器创建一个cookie的时候没有指定对应的过期时间时，客户端会将

这类cookie写入浏览器开辟的一块内存中，当关闭浏览器以后，这块内存也就被释放了，对应的cookie也就是结束了它的生命；

- 指定过期时间的cookie。当服务器创建一个cookie的时候指定了对应的过期时间时，当到达了过期时间时，对应的cookie就会被删除；

- 当浏览器中的cookie数量达到了限制时。浏览器会按照某种策略删除一些旧的cookie，腾出空间来创建新的cookie；

- 也可以人为删除cookie。

cookie管理

服务器端创建一个cookie时，一般都会指定以下两个选项：

- domain
- path

这两个选项决定了创建的cookie属于哪个域名下的哪个位置。

对于domain选项，默认情况下，domain会被设置为创建该cookie的页面所在的域名。当客户端再次给相同域名发送请求时，cookie会一起被发送至服务器。当cookie的domain选项被设置为一个一级域名时，此域名下的所有二级都将同时拥有相同的cookie，经常会出现顶级域名和二级域名的cookie冲突问题。

我们在发送请求时，浏览器会把domain的值与请求的域名做一个尾部比较（即从字符串的尾部开始比较），并将匹配的cookie发送至服务器。

- 当我们未指定domain时，默认的domain为访问地址的域名。如果是顶级域名访问，那么设置的cookie也可以被其他二级域名所共享，因此登录等操作一般都在顶级域名下进行操作。

- 二级域名可以读取设置了domain为顶级域名或者自身的cookie，但是不能读取其他二级域名domain的cookie，因此想要cookie在多个二级域名中共享的时候，需要设置domain为顶级域名，这样就可以在所有二级域名里面使用该cookie。这里需要注意的是顶级域名只能获取到domain设置为顶级域名的

版权所有：腾讯云计算（北京）有限责任公司

cookie，无法获取domain设置为二级域名的cookie。

说完domain，再来说说path选项。path选项规定，客户端请求的URL只有在存在path指定的路径时，才会发送cookie消息头，它决定了客户端发送cookie到服务器端的匹配规则。通常是将path选项的值与请求的URL从头开始逐字符比较，如果字符匹配，则发送cookie消息头。需要注意的是，只有在domain选项满足之后才会对path属性进行比较。path属性的默认值是发送Set-Cookie消息头所对应的URL中的path部分。

以上从浏览器本身的限制和生成cookie时的选项对cookie的管理进行了简单的总结。接下来就通过一些简单的代码来演示如何创建和获取cookie。

服务器端创建cookie

腾讯云服务器通过发送一个带有Set-Cookie的HTTP消息响应头来创建一个cookie。例如：

```
// 创建一个cookie对象
Cookie co = new Cookie(“site”, “http://cloud.tencent.com ”);
co.setDomain(“test.com”);
// 通过响应头，将cookie发送到客户端
response.addCookie(co);

Cookie co = new Cookie(“site”, “http://qcloud.com ”);
co.setDomain(“test.com”);
co.setPath(“/pages”);
co.setMaxAge(3600); // 单位为秒
co.setHttpOnly(true);
co.setSecure(false);
response.addCookie(co);
```

客户端读取cookie

客户端向服务器发起请求时，在domain和path匹配的情况下，会将对应的cookie一起发送到服务器端。如果一个path下设置的cookie太多，就可能出现http请求头超长的问题。请求到达服务器端以后，我们可以这样读

版权所有：腾讯云计算（北京）有限责任公司

取cookies：

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
for (int i = 0; i < cookies.length; ++i) {
// 获得具体的Cookie
Cookie cookie = cookies[i];
// 获得Cookie的名称
String name = cookie.getName();
String value = cookie.getValue();
out.print(“Cookie名:” + name + ” Cookie值:” + value + “
” );
}
}
```

HTTP返回值说明

HTTP状态码（HTTP Status Code）是用以表示网页服务器HTTP响应状态的3位数字代码。它由 RFC 2616 规范定义的，并得到RFC 2518、RFC 2817、RFC 2295、RFC 2774、RFC 4918等规范扩展。

所有状态码的第一个数字代表了响应的五种状态之一：

1xx：信息响应类，表示接收到请求并且继续处理
2xx：处理成功响应类，表示动作被成功接收、理解和接受
3xx：重定向响应类，为了完成指定的动作，必须接受进一步处理
4xx：客户端错误，客户请求包含语法错误或者是不能正确执行
5xx：服务端错误，服务器不能正确执行一个正确的请求

下面给出了常见响应返回值的说明：

100——客户必须继续发出请求
101——客户要求服务器根据请求转换HTTP协议版本

200——交易成功
201——提示知道新文件的URL
202——接受和处理、但处理未完成
203——返回信息不确定或不完整
204——请求收到，但返回信息为空
205——服务器完成了请求，用户代理必须复位当前已经浏览过的文件
206——服务器已经完成了部分用户的GET请求

300——请求的资源可在多处得到
301——删除请求数据
302——在其他地址发现了请求数据
303——建议客户访问其他URL或访问方式
304——客户端已经执行了GET，但文件未变化
305——请求的资源必须从服务器指定的地址得到
306——前一版本HTTP中使用的代码，现行版本中不再使用
307——申明请求的资源临时性删除

版权所有：腾讯云计算（北京）有限责任公司

400——错误请求，如语法错误

401——请求授权失败

402——保留有效ChargeTo头响应

403——请求不允许

404——没有发现文件、查询或URl

405——用户在Request-Line字段定义的方法不允许

406——根据用户发送的Accept拖，请求资源不可访问

407——类似401，用户必须首先在代理服务器上得到授权

408——客户端没有在用户指定的饿时间内完成请求

409——对当前资源状态，请求不能完成

410——服务器上不再有此资源且无进一步的参考地址

411——服务器拒绝用户定义的Content-Length属性请求

412——一个或多个请求头字段在当前请求中错误

413——请求的资源大于服务器允许的大小

414——请求的资源URL长于服务器允许的长度

415——请求资源不支持请求项目格式

416——请求中包含Range请求头字段，在当前请求资源范围内没有range指示值，请求也不包含If-Range请求头字段

417——服务器不满足请求Expect头字段指定的期望值，如果是代理服务器，可能是下一级服务器不能满足请求

500——服务器产生内部错误

501——服务器不支持请求的函数

502——服务器暂时不可用，有时是为了防止发生系统过载

503——服务器过载或暂停维修

504——关口过载，服务器使用另一个关口或服务来响应用户，等待时间设定值较长

505——服务器不支持或拒绝支请求头中指定的HTTP版本

版权所有：腾讯云计算（北京）有限责任公司

SSL证书链说明

1. SSL证书链定义

证书颁发机构(CA)共分为两种类型：根CA和中间CA。为了使SSL证书被信任，该证书必须由设备所连接的可信存储库CA颁发。

如果该证书不是由受信任CA，该链接设备(如网络浏览器)将检查，查看该证书是否由受信任的CA颁发，直到没有发现受信任CA为止。

SSL证书链就是证书列表中的根证书、中间证书到终端用户证书。

1. SSL证书链举例

假设用户从Qcloud机构购买证书，域名是example.qcloud.

Qcloud机构不是一个根证书颁发机构。换句话说，它的证书并不是直接嵌入在web浏览器，因此它不能被明确的信赖。

- Qcloud机构使用由中间Qcloud证书颁发机构阿尔法颁发的证书
- 中间Qcloud CA阿尔法使用由中间Qcloud证书颁发机构贝塔颁发的证书
- 中间Qcloud CA贝塔使用由中间Qcloud证书颁发机构伽马颁发的证书
- 中间Qcloud CA伽马使用由The Root of Qcloud颁发的证书
- The Root of Qcloud是一个根CA。该证书是直接嵌入在您的web浏览器中，因此可以被信任。

以上的例子中，SSL证书链是由以下6个证书组成的：

1. 终端证书：颁发给example.qcloud，发行商：Qcloud机构
2. 中间证书1：颁发给example.qcloud，发行商： 中间Qcloud证书颁发机构阿尔法
3. 中间证书2：颁发给中间Qcloud证书颁发机构阿尔法，发行商： 中间Qcloud证书颁发机构贝塔
4. 中间证书3：颁发给中间Qcloud证书颁发机构贝塔，发行商： 中间Qcloud证书颁发机构伽马
5. 中间证书4：颁发给中间Qcloud证书颁发机构伽马，发行商：The Root of Qcloud
6. 根证书： 颁发给The Root of Qcloud，由The Root of Qcloud颁发

版权所有：腾讯云计算（北京）有限责任公司

其中证书1 称为终端用户证书，证书2-5被称为中间证书。证书6被称为根证书。

当用户安装终端证书example.qcloud时，必须将所有的中间证书捆绑，并与终端证书一起安装。如果SSL证书链无效或被受损， 则用户的证书就不被某些设备信任。

版权所有：腾讯云计算（北京）有限责任公司

SSL单向认证和双向认证说明

SSL双向认证具体过程

- 浏览器发送一个连接请求给安全服务器。
- 服务器将自己的证书，以及同证书相关的信息发送给客户浏览器。
- 客户浏览器检查服务器送过来的证书是否是由自己信赖的CA中心所签发的。如果是，就继续执行协议;如果不是，客户浏览器就给客户一个警告消息：警告客户这个证书不是可以信赖的，询问客户是否需要继续。
- 接着客户浏览器比较证书里的消息，例如域名和公钥，与服务器刚刚发送的相关消息是否一致，如果是一致的，客户浏览器认可这个服务器的合法身份。
- 服务器要求客户发送客户自己的证书。收到后，服务器验证客户的证书，如果没有通过验证，拒绝连接;如果通过验证，服务器获得用户的公钥。
- 客户浏览器告诉服务器自己所能够支持的通讯对称密码方案。
- 服务器从客户发送过来的密码方案中，选择一种加密程度最高的密码方案，用客户的公钥加过密后通知浏览器。
- 浏览器针对这个密码方案，选择一个通话密钥，接着用服务器的公钥加过密后发送给服务器。
- 服务器接收到浏览器送过来的消息，用自己的私钥解密，获得通话密钥。
- 服务器、浏览器接下来的通讯都是用对称密码方案，对称密钥是加过密的。

双向认证则是需要服务端与客户端提供身份认证，只能是服务端允许的客户能去访问，安全性相对较高一些。

SSL单向认证具体过程

- 客户端的浏览器向服务器传送客户端SSL协议的版本号，加密算法的种类，产生的随机数，以及其他服务器和客户端之间通讯所需要的各种信息。
- 服务器向客户端传送SSL协议的版本号，加密算法的种类，随机数以及其他相关信息，同时服务器还将向客户端传送自己的证书。
- 客户利用服务器传过来的信息验证服务器的合法性，服务器的合法性包括：证书是否过期，发行服务器证书的CA是否可靠，发行者证书的公钥能否正确解开服务器证书的"发行者的数字签名，服务器证书的域名是否和服务器的实际域名相匹配。如果合法性验证没有通过，通讯将断开;如果合法性验证通过，将继续进行第四步。
- 用户端随机产生一个用于后面通讯的"对称密码"，然后用服务器的公钥(服务器的公钥从第二步中的服务器的证书中获得)对其加密，然后将加密后的"预主密码"传给服务器。

- 如果服务器要求客户的身份认证(在握手过程中为可选)，用户可以建立一个随机数然后对其进行数据签名，将这个含有签名的随机数和客户自己的证书以及加密过的"预主密码"一起传给服务器。
- 如果服务器要求客户的身份认证，服务器必须检验客户证书和签名随机数的合法性，具体的合法性验证过程包括：客户的证书使用日期是否有效，为客户提 供证书的CA是否可靠，发行CA的公钥能否正确解开客户证书的发行CA的数字签名，检查客户的证书是否在证书废止列表(CRL)中。检验如果没有通过，通讯立刻中断;如果验证通过，服务 器将用自己的私钥解开加密的"预主密码 "，然后执行一系列步骤来产生主通讯密码(客户端也将通过同样的方法产生相同的主通讯密码)。
- 服务器和客户端用相同的主密码即"通话密码"，一个对称密钥用于SSL协议的安全数据通讯的加解密通讯。同时在SSL通讯过程中还要完成数据通讯的完整性，防止数据通讯中的任何变化。
- 客户端向服务器端发出信息，指明后面的数据通讯将使用的上一步中的主密码为对称密钥，同时通知服务器客户端的握手过程结束。
- 服务器向客户端发出信息，指明后面的数据通讯将使用的上一步中的主密码为对称密钥，同时通知客户端服务器端的握手过程结束。
- SSL的握手部分结束，SSL安全通道的数据通讯开始，客户和服务器开始使用相同的对称密钥进行数据通讯，同时进行通讯完整性的检验。

SSL 单向认证只要求站点部署了 SSL
证书就行，任何用户都可以去访问(IP被限制除外等)，只是服务端提供了身份认证。

SSL双向认证和SSL单向认证的区别

双向认证 SSL 协议要求服务器和用户双方都有证书。单向认证 SSL 协议不需要客户拥有 CA 证书。

单向认证的具体过程相对应于上面的步骤，只需将服务器端验证客户证书的过程去掉，以及在协商对称密码方案，对称通话密钥时，服务器发送给客户的是没有加过密的(这并不影响 SSL 过程的安全性)密码方案。这样，双方具体的通讯内容，就是加过密的数据，如果有第三方攻击，获得的只是加密的数据，第三方要获得有用的信息，就需要对加密数据进行解密，这时候的安全就依赖于密码方案的安全。目前所用的密码方案，只要通讯密钥长度足够的长，就足够安全。这也是我们使用128位加密通讯的原因。

一般 Web 应用配置 SSL
单向认证即可。但部分金融行业用户的应用对接，可能会要求对客户端做身份验证。这时就需要做 SSL
双向认证。