Cloud Load Balance

More

Product Introduction

Contents

More
HTTP Long Connections

Last updated：2018-08-31 15:36:34

Currently, Tencent Cloud's cloud load balancer allows the setting of an HTTP persistent connection for Layer-7 cloud load balance to the default value of 75s, and users can customize different cloud load balancer instances. So, what is an HTTP persistent connection, and an HTTP short connection?

1. The relationship between HTTP and TCP/IP protocols

HTTP persistent connections and short connections are essentially TCP persistent connections and short connections. HTTP is used at the application layer, TCP is used at the transport layer, and IP is used at the network layer. IP protocol mainly solves the problem of network routing and addressing, and TCP protocol mainly solves the problem of how to reliably transmit data packets above the IP layer so that the receiving terminal on the network receives all the packets sent by the transmitting terminal in the transmitting order. TCP is a reliable connection-oriented protocol.

2. What does "HTTP is a stateless protocol" mean?

"HTTP is a stateless protocol" means that the protocol cannot remember transaction processing, and the server does not know the state of the client. That is to say, there is no connection between the actions of opening a web page on the same server this time and last time HTTP is a stateless connection-oriented protocol. But stateless does not mean that HTTP cannot maintain TCP connections and uses UDP protocol (no connection).

3. What is a persistent connection, and a short connection?

Short connections are used by default in HTTP/1.0. A connection is established between the client and the server for each HTTP operation, and the connection will be interrupted after a task is completed. When an HTML or other types of Web pages the client browser accesses contain other Web resources (such as JavaScript files, image files, and CSS files), the browser will establish a new HTTP session upon each encounter with such a Web resource.

For HTTP/1.1 and later versions, persistent connections are used by default to maintain connection characteristics. This line of code will be added to the response header for HTTP protocols using persistent

connections:

> **Connection**:keep-alive

If a persistent connection is used, when a web page is opened, the TCP connection between the client and the server for the transmission of HTTP data will not be closed. When the client accesses the server again, this established connection will still be used. Keep-Alive does not keep the connection permanently, and it has a persistence duration, which can be set in different server software (such as Apache). The realization of persistent connections requires support by both the client and the server.

HTTP persistent connection and short connection are essentially TCP persistent connection and short connection.

3.1 TCP connections

When a TCP protocol is used for network communication, a connection must be established between the client and the server before the read and write operations are performed. When the read and write operations are completed, the connection can be released when both sides no longer need it. The establishment of a connection depends on "three-way handshake", while the release requires "four-way handshake". So, the establishment of each connection needs to consume resources and time.

Diagram of connection establishment via three-way handshake:

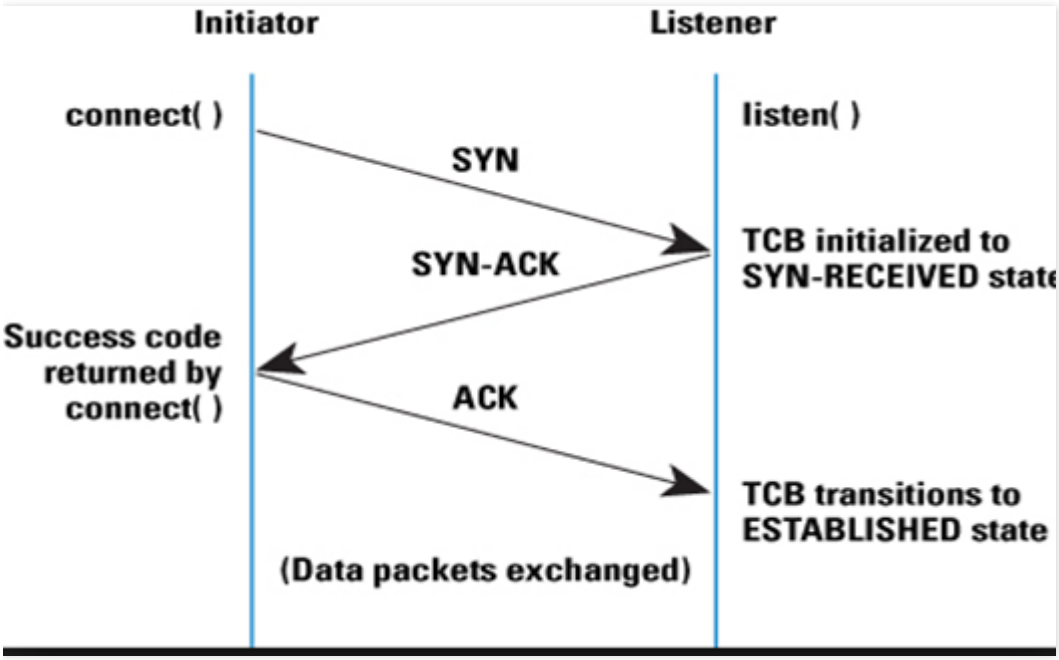
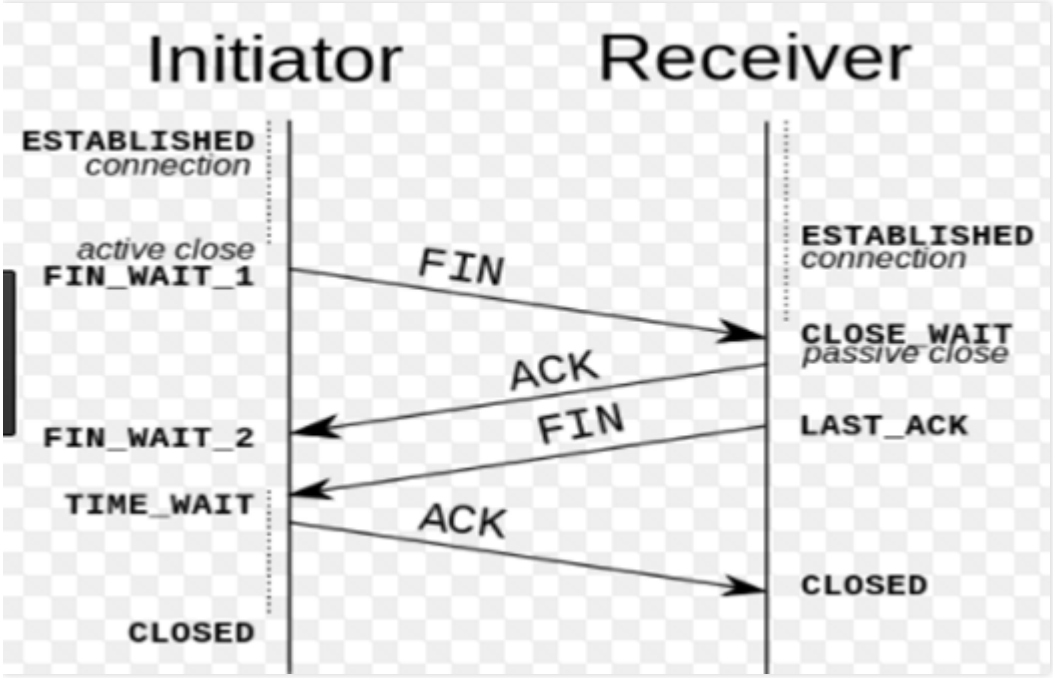


Diagram of connection closing via four-way handshake:



3.2 TCP short connections

Simulation of a TCP short connection: the client initiates a connection request to the server, the server receives the request, and then a connection between them is established. The client sends a message to the server, the server returns a response to the client, and then a request is completed. At this time, both the client and the server can initiate a close operation, but usually the client will first initiate the operation. As mentioned above, short connections only pass one request operation between the client and the server.

The advantage of short connections is that they are easy to manage, as the existing connections are all useful, and no additional control measure is needed.

3.3 TCP persistent connections

Simulation of a persistent connection: the client initiates a connection request to the server, the server accepts the request, and then a connection is established between them. After a request between the client and the server is completed, the connection between them will not be closed actively, and it will still be used for subsequent read and write operations.

TCP's keep-alive function is primarily provided for server applications. If the client has disappeared but the connection is not broken, there will be a half-open connection on the server, and the server, waiting for data from the client, will wait forever. The keep-alive function is used to detect such semi-open connections on the server.

If a given connection has not taken any actions within two hours, the server will send a detection message segment to the client, detecting the client's state based on the client's response. There are four states of the client:

- The client is running normally and the server is reachable. In this case, the client's TCP response is normal, and the server will reset the keep-alive timer.
- The client has crashed and is shut down or is rebooting. In both cases, the client cannot respond to TCP, and the server will not receive a response to the detection from the client. The server will send a total of 10 detections, with an interval of 75 seconds. If the server does not receive any responses, it will consider the client closed, and it will terminate the connection.
- The client has crashed and already rebooted. The server will receive a response to its keep-alive detection. This response is a reset to inform the server to terminate the connection.
- The client is running normally, but the server is unreachable. This is similar to the second state.

4. Advantages and disadvantages of persistent and short connections

As can be seen from the above, persistent connections can save more TCP establishment and close operations, reducing the waste of resources and time. For clients that frequently request resources, persistent connections are suitable. In the applications of persistent connections, usually, the client won't close the connection actively. If the connection between the client and the server leaves unclosed, when there are more and more connections for clients, the server will maintain too many connections. At this time, the server needs to take some measures, such as closing some connections through which no request has occurred for a long time, so as to avoid service damage on the server caused by some malicious connections. If possible, the server may limit the maximum number of persistent connections of each client, completely preventing malicious clients from wearing down the overall backend services.

For the server, short connections are easier to manage, as the existing connections are all useful, and no additional control measure is needed. But if the client makes requests frequently, plenty of time and bandwidth will be wasted for establishing and closing TCPs.

Persistent connections and short connections are established depending on the close strategy adopted by the client and the server. Different strategies should be used for different application scenarios.

WebSocket Principle

Last updated：2018-10-10 11:42:53

Web application communication is typically made as such: the client browser sends a request; the server receives the request, process it, and return the result to the client; and then the client browser displays such information. This mechanism is suitable for applications that do not need to change information frequently, but it can barely handle applications that require quick real-time response and massive concurrent requests, especially in the fast-growing mobile-Internet industry where massive concurrent requests and real-time response are the two major problems often encountered by Web applications, such as real-time information on financial securities, geo-location access in Web navigation applications and real-time messaging of social networks.

In the traditional request-response mode, Web developers often use real-time communication to deal with such business scenario. A commonly-used solution is Round Robin. That is, the client sends requests to the server at a certain time interval to synchronize data between the client and the server, which is very easy to understand. However, this solution has an obvious weak point. When the client sends a request at a fixed frequency, the data in the server may not be updated, which causes plenty of unnecessary requests, waste of bandwidth and low efficiency.

Another solution is to use Flash/AdobeFlash. It achieves real-time transmission by implementing data exchange via Socket and exposing relevant APIs to JavaScript for calling. This solution is more efficient than Round Robin, for Flash has relatively high installation rate and broad application scenarios. However, Flash is not well supported on mobile-Internet terminals. It cannot be compatible with IOS system, and does not perform well on Android system due to high requirements on hardware configurations. In 2012, Adobe announced to cease development of Flash for Android 4.1 and above, which indicates the "death" of Flash on mobile terminals.

As traditional Web modes fail to meet the requirement of massive concurrent requests and quick real-time response, the industry is in urgent need for a highly efficient and energy-saving two-way communication mechanism that can keep real-time data transmission. That's where the HTML5 WebSocket (Web TCP) comes in. There was not a standard HTML5 in early development. Each browser and application server providers had a unique but similar application, such as IBM's MQTT and Comet open source framework. It was until 2014 that a standard HTML5 was formulated and put into practice by all application server and browser providers. A standard WebSocket protocol was also implemented in JavaEE7. From then on, the WebSocket has become available on both the client and the server. For more information on the new HTML protocol and WebSocket support, refer to HTML5 Protocol.

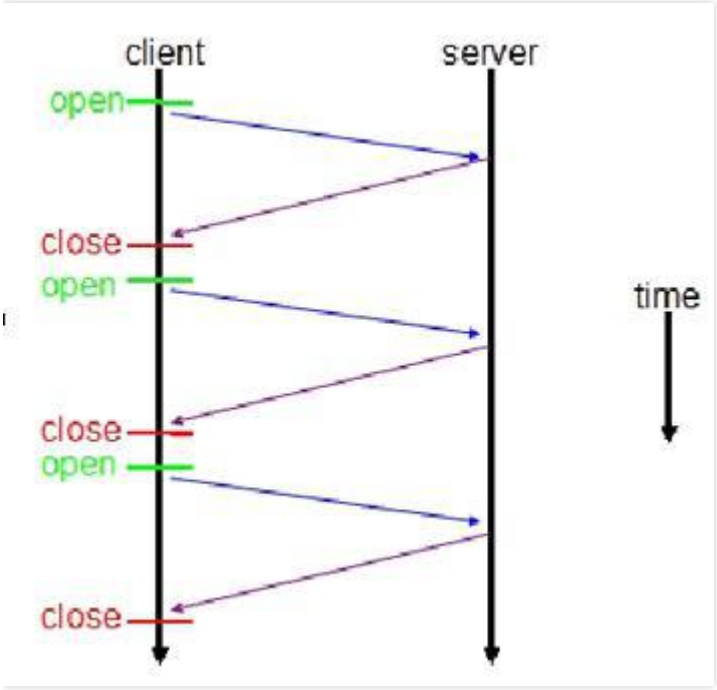
WebSocket Mechanism

The following is a brief introduction to the principle and operating mechanism of WebSocket.

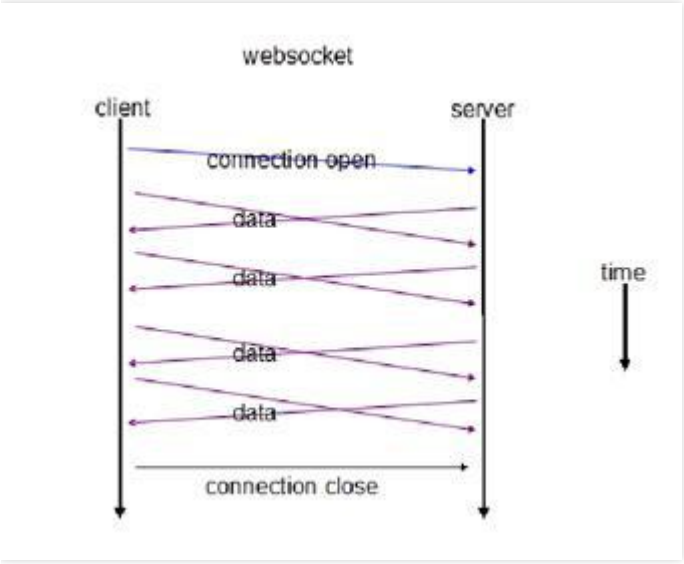
WebSocket is a new protocol under HTML5. It provides full-duplex communication between the browser and the server, which can save server resources and bandwidth and achieve real-time communication. WebSocket and HTTP both transmit data via established TCP connections. Their differences are:

- WebSocket is a two-way communication protocol. Like Socket, WebSocket server and client can send/receive data to/from each other after establishing connections;
- WebSocket, like TCP, should establish connections before communicating with each other.

Traditional HTTP client-server request-response mode is shown as follows:



WebSocket client-server request-response mode is shown as follows:



As can be seen from the figures above, traditional HTTP mode should establish connections between the client and the server for each request - response session, while WebSocket uses a TCP persistent connection communication mode similar to Socket. Once the WebSocket connection is established, subsequent data is transmitted using a sequence of frames. The client/server does not need to re-send the connection request before the client/server disconnects the WebSocket connection. When dealing with massive concurrent requests or heavy client-server interaction traffic loads, WebSocket can significantly save network bandwidth resources, which is an obvious performance advantage. In addition, it enables the client to send and receive messages from one persistent connection, demonstrating great real-timeness.

WebSocket distinguishes itself from HTTP persistent connection in the following aspects:

- WebSocket is a true full-duplex mode, in which the client and the server are equal and can send/receive requests from/to each other after establishment of connections; while the HTTP-based persistent connection is a traditional one-way mode in which only the server can send requests.
- In the HTTP persistent connection, whenever exchanging necessary data, the server and the client have to exchange a large number of HTTP headers, which leads to low data exchange efficiency; On the contrary, the WebSocket protocol does not need to exchange HTTP headers any more after the establishment of TCP connections by the first request, provided that the server and the client are updated and their browsers support HTML5. Besides, WebSocket has some new functions including multiplexing and re-use of one WebSocket connection by different URLs, which are impossible in HTTP persistent connection.

Now let's make a comparison between WebSocket and traditional HTTP based on their messages interacted between the client and the server:

The new WebSocket instantiates a new WebSocket client object on the client, and requests a WebSocket URL similar to ws://yourdomain:port/path from the server. The client WebSocket object automatically parses and recognizes the WebSocket requests, connects to the server port, and implements handshake process. The client sends data in the following format:

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

As can be seen from the above, the WebSocket connection message initiated by the client is similar to the traditional HTTP message. `Upgrade: websocket` indicates that this is a WebSocket type request. `Sec-WebSocket-Key` indicates that it is a base64-encoded message sent by the WebSocket client. The server must return a corresponding encrypted `Sec-WebSocket-Accept` response; otherwise, the client will throw the `Error during WebSocket handshake` error and close the connection.

After receiving the message, the server returns the data in the following format:

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

`Sec-WebSocket-Accept` is the value returned to the client after calculated by the server using the same key as the client. `HTTP/1.1 101 Switching Protocols` indicates that the server accepts the client connection via the WebSocket protocol. After the completion of such request-response processing, the WebSocket connection between the server and the client successfully performs handshake, and the TCP communication will be established. For more information on the format of data interacted between the client and the server via WebSocket, refer to WebSocket Protocol Stack.

It's easy to create a WebSocket API. You need to instantiate the WebSocket, establish a connection, and then the server and the client can send/respond messages to each other. You can see the detailed WebSocket API and code implementation in WebSocket Implementation and Case Study.

Tencent Cloud public network-based (with daily rate) Layer-7 CLB is capable of forwarding Websocket requests. Currently, multiple companies including Calibur of Spirit and Yinhan Games have access to such capability. Please modify the WebSocket configuration if it is not compatible with your system. WebSocket server does not check the fields encircled in the following figure:

```
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: TgxnYhlXJHsvWZbPJE1eQA8+XNo=
```

If you want to apply WebSocket in your video business, you should:

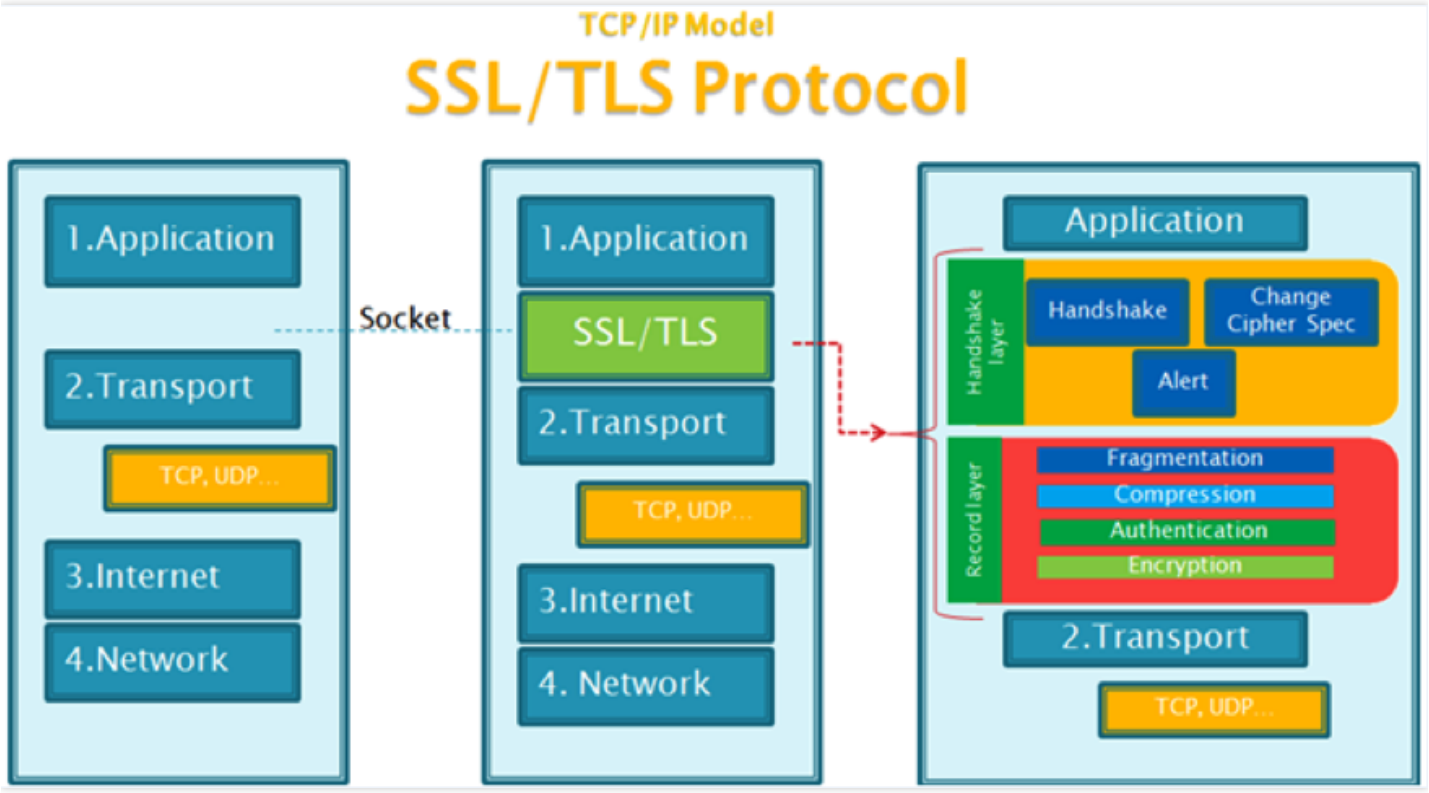
- Use heartbeat to maintain the WebSocket link to detect whether the client's Internet stars/VJs are online
- Set proxy_read_timeout of Layer-7 CLB to 60s
- Set the heartbeat to 50s, so that the WebSocket can be connected for a long time

A custom configuration section for Websocket will be available in the future.

SSL Principle

Last updated：2018-05-28 18:06:05

SSL/TLS is an optional protocol between application layer (HTTP protocol) and transport layer (TCP protocol) with an architecture as follows:



If SSL/TLS is not used in HTTP communication, all the messages will be transmitted in clear text, which will resulting in the following risks:

- Eavesdropping: The communication content is obtained by the third party
- Tampering: The communication content is changed by the third party
- Pretending: The third party takes part in the communication by use of someone else's identity

SSL/TLS protocol is developed to deal with these risks. The protocol is designed to:

- Transmit all the messages in an encrypted way to prevent the third party from eavesdropping.
- Support verification mechanism that allows the two sides to immediately discover the fact that the communication content has been tempered with.
- Avoid identity theft by providing ID certificate.

Currently, SSL/TLS is supported by most mainstream browsers.

How does SSL/TLS Protocol work

SSL/TLS protocol works based on the encryption with public key. Client sends a request for a public key to the server, then uses the public key to encrypt the message and sends it to the server; after receiving the encrypted message, the server decrypts it with its private key.

There are two issues to deal with:

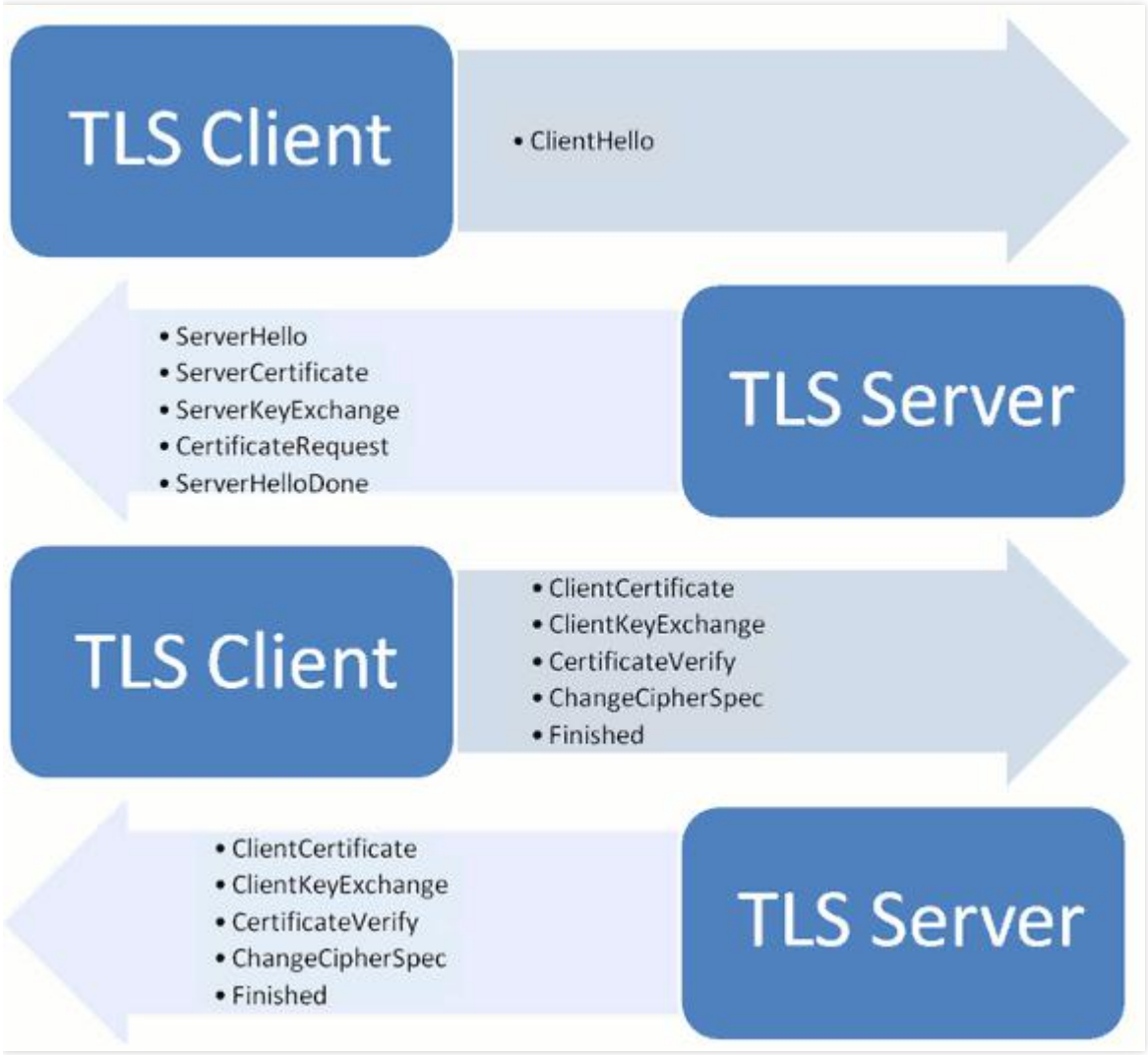
- How to prevent the public key from being tempered with?
  Solution: Put the public key into a digital certificate. As long as the certificate is credible, the public key is credible.
- How to reduce the time consumed in the heavy computing for public key encryption?
  Solution: In each session, both client and server will generate a "session key" for encryption. The "session key" uses symmetrical encryption and operates very fast. The server's public key is only used to encrypt the "session key". In this way, the time consumed in the encryption operation can be shortened.

SSL/TLS protocol works as follows:

- Client sends a request for public key to the server and verifies the public key
- The two sides generates a"session key" through negotiation
- The two sides achieves encrypted communication with the "session key".

The first two steps in the above procedure are known as "handshake".

Work Flow of Handshake



"Handshake" involves four communications, with all the messages transmitted in clear text.

Request from Client (ClientHello)

The client (usually a browser) sends a request for encrypted communication to the server. The request is commonly known as ClientHello.

In this step, the client mainly provides the following information to the server.

- Supported protocol version, e.g. TLS 1.0
- A random number generated by the client, which will be used to generate "session key" later.
- Supported encryption method, e.g. RSA public key encryption
- Supported compression method

Please note that the message sent from the client does not contain server's domain. That is to say, the server can only have one website in theory, otherwise it can be confused about which website's digital certificate should be provided for the client. This is why a server can only have one digital certificate.

However, this is inconvenient for the users of virtual host. In 2006, TLS protocol was supplemented with a Server Name Indication Extension to allow the client to provide the server with the domain name it requests.

Response from Server (SeverHello)

After receiving the request from the client, the server sends a response to the client, which is commonly known as "SeverHello". The response includes the following information:

- The encrypted communication protocol version to be used, such as TLS 1.0. If the version supported by the browser is different from that by the server, the server will disable the encrypted communication
- A random number generated by the server, which will be used to generate "session key" later
- The encryption method to be used, e.g. RSA public key encryption
- Server certificate

In addition to the above information, the server may send another request for the "server certificate" to the client if the sever needs to verify the client's identity. For example, a financial institution only permits the verified customers to connect to their network, so they will provide a USB key containing a client certificate to the regular customers.

Response from Client

After receiving the response from the server, the client will verify the server certificate first. If the certificate is not issued from a trusted certificate authority (CA), the domain name in the certificate is different from the actual domain name, or the certificate is expired, the visitor will receive a warning message that allows the visitor to select whether to continue the communication.
If no problem is found with the certificate, the client will get the public key from the certificate and send the following information to the server:

- A random number that will be encrypted with the server public key to avoid being eavesdropped.
- Encoding change notification that indicates that the messages transmitted since now will be sent using the encryption method and the key agreed by both sides.
- Client handshake termination notification that indicates that handshake stage on client has ended. This is also the hash value for all the content sent in the previous steps and is used by the server for verification.

The random number here is the third random number during the handshake stage, which is also called "pre-master key". With this number, both client and server have three random numbers, and then each of them will use the agreed encryption method to generate the same "session key" used in this session.

For the RSA key exchange algorithm, pre-master-key itself is a random number, which together with the random numbers in hello messages will be exported as a symmetric key by a key exporter.

The reason why pre-master exists is that SSL protocol doesn't consider that each host can generate a totally random number. The key generated from the three random numbers (the ones generated from client and server, plus pre-master) is unlikely to be cracked. A pseudo-random number may not be totally random, but three ones are nearly random.

In addition, if the server requests for the client certificate in the previous step, the client will send the certificate and relevant information in this step.

Final Response from Server

After receiving the third random number "pre-master-key" from the client, the server will generate the "session key" used in this session, and then send the following information to the client:

- Encoding change notification that indicates that the messages transmitted since now will be sent using the encryption method and the key agreed by both sides.
- Server handshake termination notification that indicates that handshake stage on server has ended. This is also the hash value for all the content sent in the previous steps and is used by the client for verification.

Now, the handshake stage ends completely. Next, the client and server start the encrypted communication by using HTTP protocol, with the communication content encrypted with the "session key".

A simple example can be given here to illustrate this. If A (SSL client) communicates with B (SSL server), the encrypted messages will be enclosed in [] to highlight the difference between these messages and the messages in clear text. The description of the actions performed by both sides will be enclosed in parenthesis "()".

- A:

> "I want to communicate with you securely. I have the symmetric encryption algorithms DES and RC5, key exchange algorithms RSA and DH, and digest algorithms MD5 and SHA."

- B:

> "Let's **use** the combination **of** DES, RSA **and** SHA. This **is** my certificate **with** my **name and public** key. You can **use** it **to verify** my identity."

Send the certificate to A.

> "That is all."

- A:

Check whether B's name is correct in the certificate, and verify the authenticity of B's certificate with an available CA certificate. If one of them is wrong, A will send a warning and disconnect with B. This step can ensure the authenticity of B's public key.

Generate a secret message, which will be used as an encrypted key after being processed to encrypt initialization vector (IV) and the key for hmac. Encrypt this secret message (called per_master_secret in protocol) with B's public key, and encapsulate it into a message called ClientKeyExchange. The use of B's public key can prevent the third party from eavesdropping.

> "I have generated a secrete message and encrypted it with your public key. Here it is." Send ClientKey Exchange to B. "Please note that I will send messages to you in an encrypted way!"

Process the secret message, generate an encryption key, and encrypt IV and the key for hmac.

> [Over.]

- B:

Use its own key to decrypt the secret message in ClientKeyExchange, and then process this message, generate an encryption key, and encrypt VI and the key for hmac. Now, the two sides have agreed on a set of encryption methods securely.

> "Please note that I will also send messages to you in an encrypted way!"
>
> [Over.]

- A:

> [My secret is...]

- B:

> [Others will never know this secret...]

Session Persistence Principle

Last updated：2017-12-12 10:45:45

What is Session Persistence?

Session persistence is one of the most common issues with cloud load balancer, and it is also a relatively complex one. Session persistence is sometimes called Sticky Sessions. Session persistence refers to a mechanism on the cloud load balancer that can identify the association between the client and the CVM during interaction, ensuring that a series of associated access requests keep allocating to a single server while performing cloud load balance.

When do I need session persistence?

Before discussing this problem, we must take the time to figure out some concepts: what is a Connection, what is a Session, and the difference between the two. It is important to note that if we are only talking about cloud load balance, sessions and connections usually have the same meaning.

From a simple perspective, if the user needs to log in, then you can simply regard it as a session; if the user does not need to log in, then it is a connection.

For packets in the same connection, cloud load balancer will perform NAT translation before forwarding them to a specific backend CVM for processing. The cloud load balance system contains a specific table to record the status of these connections, including "Source IP: Port", "Destination IP: Port", "Server IP: Port", Idle Timeout, and so on. Because the table recording the connection status internally on cloud load balancer consumes the system's memory, it cannot be infinitely large, and all traditional manufacturers will face some restrictions. The size of this table is generally called the maximum number of concurrent connections, that is, how many connections the system can accommodate at the same time. In items of the current connection status table on cloud load balancer a parameter of Idle Timeout is designed. When a connection has no traffic during Idle Timeout, the cloud load balancer automatically deletes this connection entry and frees the system resources.

After the connection is deleted, the client's request will not certainly be sent to the same backend CVM; instead, the CLB's traffic distribution policy prevails.

In some situations where login is required, a session is required between the client and the server to record various information on the client. For example, in most e-commerce applications or online systems that require user authentication, a client often go through several interactions with the server to complete a transaction or a request. Since these interactions are closely interrelated, in a certain interactive step of

these interactions, the server often needs to know the result(s) from the last or last few interactions, which in turn requires that all of these associated interactions be completed by one server and cannot be distributed by the cloud load balancer to a different server. Otherwise, abnormalities may occur:

- The client entered the correct user name and password, but was repeatedly redirected to the login page;
- The user entered the correct verification code, but was always alerted to verification code error;
- Items that the client puts into the shopping cart were lost
- ...

The significance of the session persistence mechanism is therefore to ensure that requests from the same client are forwarded to the same backend CVM for further processing. In other words, the connections established between the client and the server are sent to the same server for processing. If a cloud load balancer is deployed between the client and the server, it is likely that the multiple connections will be forwarded to different servers for processing. If there is no session information synchronization mechanism between servers, other servers may not identify the user's identity, resulting in exception on interaction between the user and the application system.

Cloud load balancer is to have the connections and requests from clients evenly forwarded to multiple CVMs to avoid overload of a single CVM; while the session persistence mechanism requires that certain requests be forwarded to the same server. Therefore, in the actual deployment environment, we have to select the appropriate session persistence mechanism according to the characteristics of the application environment.

Session Persistence Category

Simple Session Persistence (Layer-4 Session Persistence)

Simple Session Persistence (also known as source address-based session persistence or IP based session persistence) means that the cloud load balancer uses the source address of the access request as the basis for determining the associated session. All access requests from the same IP address are delivered to a same CVM during cloud load balancing.

A very important parameter in a simple session persistence is the connection timeout value; the cloud load balancer sets a time value for each session that is in a persistent state. If the interval of a session from the last completion to the next iteration is less than the timeout value, the cloud load balancer will have the new connection persisted for the same session; but if the interval is greater than the timeout value, the cloud load balancer will regard the new connection as a new session and perform cloud load balancing.

It's easy to implement simple session persistence, which requires only layer-3 or 4 information of packets, the efficiency being high.

NginX can support simple session persistence
ip_hash
Requests are distributed according to hash results of accesing IPs, ensuring that each visitor can regularly visit one RS so as to solve the problem of session.
For example:

```
upstream bakend {
ip_hash;
server 192.168.0.14:88;
server 192.168.0.15:80;
}
```

However, the problem with this approach is that when multiple clients access the server through proxy or address translation, the requests are delivered to the same server due to the same source address, resulting in a serious imbalance between servers.

In another situation, when a same client generates a large number of concurrent requests to multiple servers for further processing, session persistence occurs. In this case, the source address based session persistence will also lead the cloud load balancer to failure.

In above situations, we must consider using other ways for session persistence.

Session Saving Session Persistence

This method implements session persistence during cloud load balancing by sharing session between multiple backend servers. It mainly takes the following forms:

1) Database Storage

Session information is stored in the database table to achieve session information sharing between different application servers. This method is suitable for sites with few database access.

- Advantages: Easy to implement
- Disadvantages: Because the database server is more expensive and more difficult to scale out than the application server, For Web applications in high concurrency, the biggest performance bottleneck is usually in the database server. So if session is stored in the database table, you service may be affected due to frequent database operations.

2) File System Storage

Sharing session between various servers is achieved through a file system (such as NFS). This method is suitable for sites in low concurrency.

- Advantages: For each server, only disks saving the Session are to mount, which is easy to implement.
- Disadvantages: NFS can't provide high performance for high concurrent read and write, and has huge bottleneck in the hard disk I/O performance and network bandwidth, especially in frequent read from and write to such small file as Session.

3) Memcached Storage

Use Memcached to save Session data which is read directly from the memory.

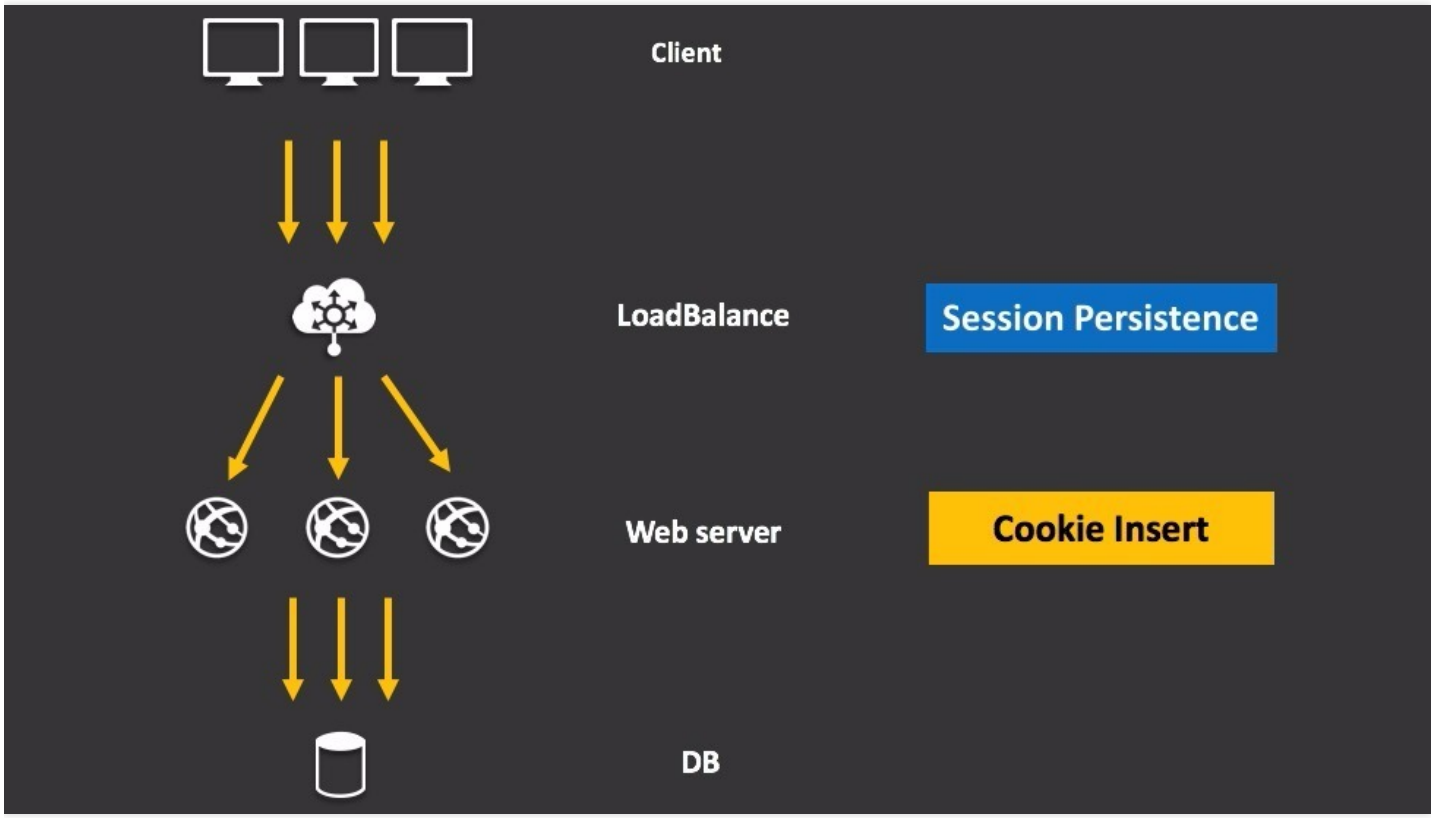
- Advantages: high efficiency, as read and write will be much faster than that stored in a file system, and sharing Session between multiple servers is more convenient; these servers are simply configured to use the same group memcached server, reducing the additional workload.
- Disadvantages: Data in the memory will be lost in case of a crash, but it is not a big deal for the Session data. If the site has very large traffic which causes too many sessions, memcached will delete the uncommonly used part. However, if the user continues to use after a period of time, read failure will occur.

Cookie-based Session Persistence (Layer-7 Session Persistence)

In the cookie based mode, CLB is responsible for inserting cookies without making any modification to backend CVM. When the client makes the first request, the client HTTP request (without cookie) is directed to CLB, which then selects a backend CVM basing on load balancing algorithm policy and sends the request to the CVM. Then CVM's HTTP response (without cookie) is sent back to CLB. After that, the CLB inserts a cookie into the backend CVM and returns the HTTP response to the client.

When the client makes the second request, the client HTTP request containing the cookie inserted by CLB last time is directed to CLB, which then reads the session persistence values in the cookie and sends the HTTP request (with the same cookie as above) to the specified CVM. Then the backend CVM gives a response, and because the CVM does not write the cookie, the HTTP response does not contain the cookie. When the HTTP response flows into the CLB again, the updated session persistence cookie will be written to CLB.

Tencent Cloud CLB layer 7 session persistence is implemented exactly basing on such a cookie insertion method.



cookie Principle

Last updated：2017-12-12 10:52:02

What Is a Cookie?

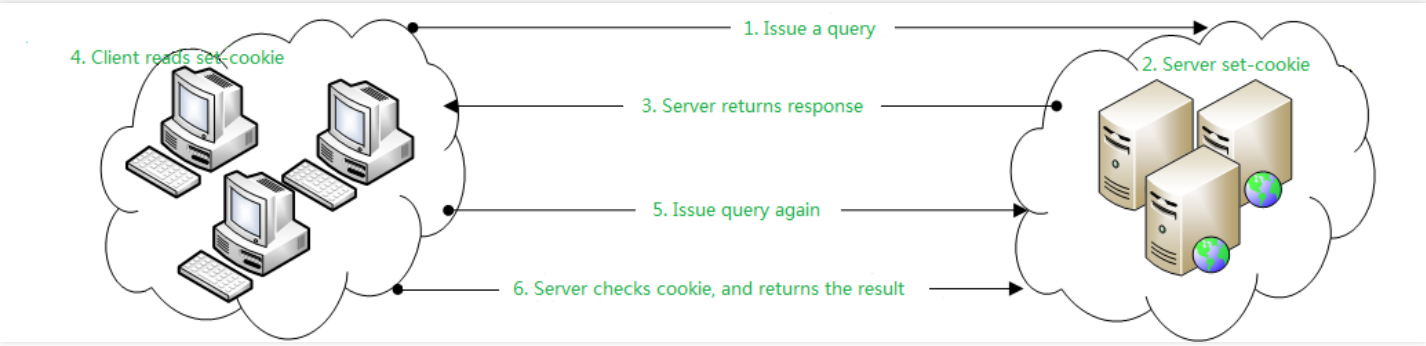
HTTP is a stateless protocol, which means that the client and the server do not need to establish a persistent connection. The connection between the client and the server is based on a request-response mode: the client and the server establish a connection - the client submits a request - the server returns a response after receiving a request, and then they are disconnected.

So, if the client and the server will be disconnected and no longer have any relationship after a request is completed, when a user logs in on page 1 and then is navigated to page 2 of the same Web application, how does page 2 know that the user has already logged in? In other words, when the client initiates another request, how does the server determine whether the two different requests are from the same client?

Under HTTP protocols, the server is unable to identify the relationship between requests. To determine the relationship, a status is necessary to identify each request. If the status IDs of two requests are the same, it means that the two requests are initiated from the same client.

Cookie is a status bit used to identify each request. After years of development, cookie becomes more and more standardized, and has been regarded as a universal standard.

How Do Cookies Work?



1) When a request is initiated to Tencent Cloud for the first time, the HTTP request header is as follows:

```
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) After the request reaches Tencent Cloud server, Tencent Cloud server will generate a response, and write cookie information in the response header:

```
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 1
4:14:50 GMT; domain=www.qcloud; path=/
Set-Cookie:BDSVRTM=172; path=/
```

3) After receiving the response header, the client browser will write the cookie information to the local for management.

4) When initiating another request to the server, the client will send a HTTP request header containing Cookie: name=value; name2=value2 together with the cookie stored in the local previously. The request header information includes:

```
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=15071962345319158759570 57
Host:cloud.tencent.com
```

5) After receiving the request, the server will get the cookie information from the request header, analyze the cookie data and then return a response to the client.

This is how cookies are used to transmit information between the client and the server.

Cookie Lifetime

Cookies have a lifetime. Once the expiration date is reached, the cookie on the client will be deleted. The server can determine how long a cookie can "survive" on the client when creating the cookie. The cookie will be ended in the following cases:

- Cookies for which the expiration time is not specified. If the server does not specify the expiration time of a cookie when creating a cookie, the client will write such cookie in a chunk of memory opened by

the browser. When the browser is closed, the memory will be freed, and the corresponding cookie will be ended;

- Cookies for which the expiration time is specified. If the server specifies the expiration time when creating a cookie, when the expiration time is reached, the corresponding cookie will be deleted;
- The number of cookies in the browser reaches the limit. The browser will delete some old cookies by a certain policy to make room for new cookies;
- You can also delete cookies by yourself.

Cookie Management

When creating a cookie, the server will generally specify the following two options:

- domain
- path

These two options determine the domain and location of the created cookie.

By default, "domain" will be set to the domain under which the page creating the cookie is located. When the client sends a request to the same domain again, the cookie will be sent to the server. When the "domain" of a cookie is set to a first-level domain, all the second-level domains under this domain will have the same cookie. There are often cookie conflicts between the top-level and second-level domains.

When a request is sent, the browser will make an end comparison of the "domain" value and the requested domain (that is, compare the domains from the end of string), and send the matched cookie to the server.

- When not specified, "domain" is the domain of the access address by default. If it is a top-level domain access, the cookie set can also be shared by other second-level domains. So login and other operations are generally performed under the top-level domain.
- Second-level domains can read cookies that set "domain" to a top-level domain or to themselves, but cannot read the cookies of other second-level domains. Therefore, if you want to share a cookie among multiple second-level domains, you need to set "domain" to a top-level domain, and then, you can use the cookie in all second-level domains. One thing to note here is that the top-level domain can only get the cookies that set "domain" to a top-level domain, but cannot get the cookies that set "domain" to a second-level domain.

As for the "path", it stipulates that a cookie message header can be sent for the URL requested by the client only when there is a path specified by "path". It determines the matching rule for the client to send a cookie to the server. Generally, the "path" value will be compared with the requested URL from the beginning by character, and if the characters are matched, the cookie message header will be sent. It should be noted that the "path" attribute will be compared only if the "domain" option is satisfied. The default value for the "path" attribute is to send the path part of the URL corresponding to the Set-Cookie message header.

The above is a summary for the management of cookies with respect to the limitations of browsers and the options for generating cookies. Next, we will show you how to create and get cookies with some simple codes.

The server creates a cookie

The Tencent Cloud server creates a cookie by sending an HTTP message response header with Set-Cookie. For example:

```
// Create a cookie object
Cookie co = new Cookie("site", "http://cloud.tencent.com");
co.setDomain("test.com");
// Send the cookie to the client via the response header
response.addCookie(co);

Cookie co = new Cookie("site", "http://qcloud.com");
co.setDomain("test.com");
co.setPath("/pages");
co.setMaxAge(3600); // In seconds
co.setHttpOnly(true);
co.setSecure(false);
response.addCookie(co);
```

The client reads the cookie

When the client initiates a request to the server, it will also send corresponding cookies to the server if the "domain" and "path" are matched. If there are too many cookies under the same "path", the http request header may be overlong. After the request reaches the server, we can read cookies like this:

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
for (int i = 0; i < cookies.length; ++i) {
// Get a specific cookie
Cookie cookie = cookies[i];
// Get the name of the cookie
```

```
String name = cookie.getName();
String value = cookie.getValue();
out.print("Cookie name: " + name + " Cookie value:" + value + "
");
}
}
```

HTTP Returned Codes

Last updated：2017-03-24 16:17:23

HTTP Status Code is a three-digit code used to indicate the web server HTTP response status. It is defined by RFC 2616 and is extended by RFC 2518, RFC 2817, RFC 2295, RFC 2774, RFC 4918 and other specifications.

The first digit of the status code specifies one of five standard classes of responses.

1xx: Informational responses, indicating that the request has been received and needs to be processed further
2xx: Success responses, indicating that the action has been successfully received, understood and accepted
3xx: Redirection responses, indicating that in order to complete the specified action, further processing is required
4xx: Client errors, indicating that the client request contains syntax errors or cannot be executed correctly
5xx: Server errors, indicating that the server cannot execute a correct request correctly

Here are descriptions of common response return values:

100 - The client must continue to send the request
101 - The client requests the server to switch HTTP protocol versions as requested

200 - The transaction is successful
201 - Prompt that the URL of the new file has been known
202 - The request has been accepted for processing, but the processing has not been completed.
203 - The returned message is indefinite or incomplete
204 - The request has been received, but the returned message is empty
205 - The server has completed the request, but the user agent must reset the viewed files
206 - The server has completed part of the user's GET requests

300 - There are multiple options for the requested resource
301 - The request data was deleted
302 - The request data was found at other addresses
303 - The client is recommended to visit other URLs or use other access methods
304 - The client has executed a GET, but the file has not changed
305 - The requested resource must be obtained from the address specified by the server
306 - The code used in the previous version of HTTP is no longer used in the current version
307 - Declaring that the requested resource was temporarily deleted

400 - Bad request, such as syntax error
401 - The request authorization failed
402 - The valid ChargeTo header response is kept
403 - The request is forbidden
404 - No file, query, or URL was found
405 - The method defined by the user in the Request-Line field is not allowed
406 - The requested resource is not accessible according to the Accept header sent by the user
407 - Similar to 401, the user must be authorized first on the proxy server
408 - The client failed to complete the request within the time specified by the user
409 - The request cannot be completed in view of the current resource status
410 - This resource is no longer available on the server and there is no reference address
411 - The server rejects the user-defined Content-Length attribute request
412 - One or more request header fields are incorrect in the current request
413 - The requested resource is larger than the size allowed by the server
414 - The requested resource URL is longer than the length allowed by the server
415 - The format of the requested resource is not supported
416 - The request contains the Range request header field, but there is no range indication value within the current requested resource, and the request does not contain the If-Range request header field
417 - The server does not satisfy the expected value specified in the request Expect header field. if it is a proxy server, it's possible that the next level server cannot meet the request

500 - An internal server error occurred
501 - The requested function is not supported by the server
502 - The server is temporarily unavailable, sometimes to prevent system overload
503 - The server is overloaded or down for maintenance
504 - The gateway is overloaded, and the server uses another gateway or service to respond to the user, but the waiting time set is too long
505 - The server does not support or reject the HTTP version specified in the request header

SSL Certificate Chain

Last updated：2017-03-28 10:52:27

What is the SSL Certificate Chain?

There are two types of CAs: Root CAs and intermediate CAs. In order for an SSL certificate to be trusted, that certificate must have been issued by a CA that is included in the trusted store of the connecting device.

If the certificate was not issued by a trusted CA, the connecting device (e.g. a web browser) will then check if the certificate was issued by a trusted CA until no trusted CA can be found.

The list of SSL certificates, from the root certificate, to intermediate certificate, to the end-user certificate, represents the SSL certificate chain.

Example of an SSL Certificate chain

Let's suppose that you purchase a certificate from the Tencent Cloud Authority for the domain example.tencent-cloud.

Tencent Cloud Authority is not a root certificate authority. In other words, its certificate is not directly embedded in your web browser and therefore it can't be explicitly trusted.

- Tencent Cloud Authority uses a certificate issued by the intermediate Tencent Cloud CA - Alpha.
- Intermediate Tencent Cloud CA Alpha uses a certificate issued by the intermediate Tencent Cloud CA - Beta.
- Intermediate Tencent Cloud CA Beta uses a certificate issued by the intermediate Tencent Cloud CA - Gamma.
- Intermediate Tencent Cloud CA Gamma uses a certificate issued by The Root of Tencent Cloud.
- The Root of Tencent Cloud is a Root CA. Its certificate is directly embedded in your web browser, therefore it can be explicitly trusted.

In the above example, the SSL certificate chain is made up of 6 certificates:

1. End-user certificate - Issued to: example.tencent-cloud; Issued By: Tencent Cloud Authority
2. Intermediate certificate 1 - Issued to: example.tencent-cloud; Issued by: Intermediate Tencent Cloud CA, Alpha

3. Intermediate certificate 2 - Issued to: intermediate Tencent Cloud CA, Alpha; Issued by: Intermediate Tencent Cloud CA, Beta
4. Intermediate certificate 3 - Issued to: intermediate Tencent Cloud CA, Beta. Issued by: Intermediate Tencent Cloud CA, Gamma
5. Intermediate certificate 4 - Issued to: intermediate Tencent Cloud CA, Gamma. Issued by: The Root of Tencent Cloud
6. Root certificate: Issued by and to: The Root of Tencent Cloud

Certificate 1 is your end-user certificate, and certificates from 2 to 5 are called intermediate certificates. Certificate 6 is root certificate.

When you install your end-user certificate for example.tencent-cloud, you must bundle all the intermediate certificates and install them along with your end-user certificate. If the SSL certificate chain is invalid or broken, your certificate will not be trusted by some devices.

SSL Mutual/Unidirectional Authentication

Last updated：2017-03-28 10:52:42

How does SSL mutual authentication work

- The browser sends a connection request to the secure server.
- The server sends its certificate along with the certificate-related information to the client browser.
- The client browser checks whether the certificate sent from the server is issued by the CA center it trusts. If so, it will continue performing the protocol; otherwise, the client browser will send the user a warning message indicating this certificate is not credible and asking whether it wants to continue.
- Next, the client browser checks whether the information (such as domain name and public key) in the certificate is consistent with the relevant information sent by the server just now. If so, the client browser will recognize the authenticity of this server's identity.
- The server requests the client to send its own certificate. After receiving the certificate, the server verifies the client's certificate. If the verification fails, the server will reject the connection request; otherwise, the server will get the user's public key.
- The client browser sends the server a message indicating the communication symmetric encryption schemes it supports.
- The server selects an encryption scheme with the highest encryption level from the encryption schemes sent from the client, and sends the scheme that is encrypted with client's public key to the browser.
- The browser selects a session key for this encryption scheme, then uses server's public key to encrypt the session key and sends it to the server.
- When receiving the message from the browser, the server decrypts the message with its private key to get the session key.
- The subsequent communications between the server and browser will be performed using the symmetric encryption scheme with the encrypted symmetric key.

Mutual authentication means that the server and client authenticate each other and the server can only be accessed by the clients permitted by the server. This can result in a much higher security.

How does SSL unidirectional authentication work

- The client browser sends the server the version number of SSL protocol, type of encryption algorithm, generated random numbers, and all the information needed in the communication between the server and client.

- The server sends the client the version number of SSL protocol, type of encryption algorithm, generated random numbers, and other relevant information as well as its certificate.
- The client uses the information sent from the server to verify the validity of the server, including whether the certificate is expired, whether the CA issuing the server certificate is credible, whether the public key of CA's certificate can correctly decrypt the CA's digital signature in the server certificate, and whether the domain name in the server certificate matches the server's actual domain name. If the validity verification fails, the communication will be disconnected; otherwise go to step 4.
- The client randomly generates a "symmetric key" for the communication later, and uses server's public key (obtained from the server certificate in step 2) to encrypt it, and then sends the encrypted "pre-master key" to the server.
- If the server requests for an authentication against the client (optional in handshake), the user can generate a random number which is then signed digitally, and sends the signed random number, along with client certificate as well as the encrypted "pre-master key" to the server.
- If the server requests for an authentication against the client, it must verify the validity of client certificate and signed random number, including: whether the certificate is expired, whether the CA issuing the certificate to the client is credible, whether the CA's public key can correctly decrypt the CA's digital signature in the client certificate, and whether the client certificate is in the Certificate Revocation List (CRL). If the verification fails, the communication will be disconnected immediately; otherwise, the server will use its own private key to decrypt the encrypted "pre-master key", and then perform a series of steps to generate a master key (the client will generate the same master key in the same way).
- The server and the client use the same master key (i.e. "session key"), a symmetric key, to encrypt and decrypt the SSL protocol-based secure data communication. Meanwhile, during the SSL communication, both sides need to make sure the integrity of data communication and avoid any changes in data communication.
- The client sends the server a message indicating that the master key in the previous step will be used as a symmetric key in the subsequent data communications and that the handshake at the client side has ended.
- The server sends the client a message indicating that the master key in the previous step will be used as a symmetric key in the subsequent data communications and that the handshake at the server side has ended.
- With the ending of handshake of SSL communication, the data communication in SSL secure channel starts. The client and the server start to use the same symmetric key for data communication while verifying the integrity of the communication data.

SSL unidirectional authentication only requires the SSL certificate to be deployed on the site. Any user can access the site (except for the restricted IPs) and only the server needs to provide the authentication information.

Difference between SSL Mutual Authentication and SSL Unidirectional Authentication

In SSL mutual authentication, both server and client are required to have certificates, while in SSL unidirectional authentication, CA certificate is not required on the client side.

Compared to the above steps, unidirectional authentication only eliminates the step in which the server verifies the client certificate. In addition, during the negotiation concerning symmetric encryption scheme and symmetric session key between the two sides, the encryption scheme sent from the server to the client s unencrypted (this will not affect the security of SSL procedure). In this way, the communication content between the two sides are encrypted data, and any third party launching a attack only obtains the encrypted data. The third party needs to decrypt the encrypted data to get useful information, and in this case, the security hinges on the security of encryption scheme. The encryption schemes available at present are sufficiently secure as long as the communication key is long enough. This is why we use 128-bit encrypted communication.

Generally, Web applications are simply configured with SSL unidirectional authentication. For some users in financial industry, an authentication against the client may be required for their application interfacing. In this case, SSL mutual authentication is needed.