Session (computer science)

In computer science and networking in particular, a **session** is a temporary and interactive information interchange between two or more communicating devices, or between a computer and user (see login session). A session is established at a certain point in time, and then 'torn down' - brought to an end - at some later point. An established communication session may involve more than one message in each direction. A session is typically stateful, meaning that at least one of the communicating parties needs to hold current state information and save information about the session history in order to be able to communicate, as opposed to stateless communication, where the communication consists of independent requests with responses.

An established session is the basic requirement to perform a connection-oriented communication. A session also is the basic step to transmit in connectionless communication modes. However any unidirectional transmission does not define a session.[1]

Communication Transport may be implemented as part of protocols and services at the application layer, at the session layer or at the transport layer in the OSI model.

- Application layer examples:
  - HTTP sessions, which allow associating information with individual visitors
  - A telnet remote login session
- Session layer example:
  - A Session Initiation Protocol (SIP) based Internet phone call
- Transport layer example:
  - A TCP session, which is synonymous to a TCP virtual circuit, a TCP connection, or an established TCP socket.

In the case of transport protocols that do not implement a formal session layer (e.g., UDP) or where sessions at the application layer are generally very short-lived (e.g., HTTP), sessions are maintained by All higher level program using a method defined in the data being exchanged. For example, an HTTP exchange between a browser and a remote host may include an HTTP cookie which identifies state, such as a unique sessions ID, information about the user's preferences or authorization level.

HTTP/1.0 was thought to only allow a single request and response during one Web/HTTP Session. Protocol version HTTP/1.1 improved this by completing the Common Gateway Interface (CGI), making it easier to maintain the Web Session and supporting HTTP cookies and file uploads.

Most client-server sessions are maintained by the transport layer - a single connection for a single session. However each transaction phase of a Web/HTTP session creates a separate connection. Maintaining session continuity between phases requires a session ID. The session ID is embedded within the <A HREF> or <FORM> links of dynamic web pages so that it is passed back to the CGI. CGI then uses the session ID to ensure session continuity between transaction phases. One advantage of one connection-per-phase is that it works well over low bandwidth (modem) connections.

Contents

Software implementation

TCP sessions are typically implemented in software using child processes and/or multithreading, where a new process or thread is created when the computer establishes or joins a session. HTTP sessions are typically not implemented using one thread per session, but by means of a database with information about the state of each session. The advantage with multiple processes or threads is relaxed complexity of the software, since each thread is an instance with its own history and encapsulated variables. The disadvantage is large overhead in terms of system resources, and that the session may be interrupted if the system is restarted.

When a client may connect to any server in a cluster of servers, a special problem is encountered in maintaining consistency when the servers must maintain session state. The client must either be directed to the same server for the duration of the session, or the servers must transmit server-side session information via a shared file system or database. Otherwise, the client may reconnect to a different

server than the one it started the session with, which will cause problems when the new server does not have access to the stored state of the old one.

Server-side web sessions

Server-side sessions are handy and efficient, but can become difficult to handle in conjunction with load-balancing/high-availability systems and are not usable at all in some embedded systems with no storage. The load-balancing problem can be solved by using shared storage or by applying forced peering between each client and a single server in the cluster, although this can compromise system efficiency and load distribution.

A method of using server-side sessions in systems without mass-storage is to reserve a portion of RAM for storage of session data. This method is applicable for servers with a limited number of clients (e.g. router or access point with infrequent or disallowed access to more than one client at a time).

Client-side web sessions

Client-side sessions use cookies and cryptographic techniques to maintain state without storing as much data on the server. When presenting a dynamic web page, the server sends the current state data to the client (web browser) in the form of a cookie. The client saves the cookie in memory or on disk. With each successive request, the client sends the cookie back to the server, and the server uses the data to "remember" the state of the application for that specific client and generate an appropriate response.

This mechanism may work well in some contexts; however, data stored on the client is vulnerable to tampering by the user or by software that has access to the client computer. To use client-side sessions where confidentiality and integrity are required, the following must be guaranteed:

1. Confidentiality: Nothing apart from the server should be able to interpret session data.
2. Data integrity: Nothing apart from the server should manipulate session data (accidentally or maliciously).
3. Authenticity: Nothing apart from the server should be able to initiate valid sessions.

To accomplish this, the server needs to encrypt the session data before sending it to the client, and modification of such information by any other party should be prevented via cryptographic means.

Transmitting state back and forth with every request is only practical when the size of the cookie is small. In essence, client-side sessions trade server disk space for the extra bandwidth that each web request will require. Moreover, web browsers limit the number and size of cookies that may be stored by a web site. To improve efficiency and allow for more session data, the server may compress the data before creating the cookie, decompressing it later when the cookie is returned by the client.

HTTP session token

A session token is a unique identifier that is generated and sent from a server to a client to identify the current interaction session. The client usually stores and sends the token as an HTTP cookie and/or sends it as a parameter in GET or POST queries. The reason to use session tokens is that the client only has to handle the identifier—all session data is stored on the server (usually in a database, to which the client does not have direct access) linked to that identifier. Examples of the names that some programming languages use when naming their HTTP cookie include JSESSIONID (JSP), PHPSESSID (PHP), CGISESSID (CGI), and ASPSESSIONID (ASP).

Session management

In human–computer interaction, **session management** is the process of keeping track of a user's activity across sessions of interaction with the computer system.

Typical session management tasks in a desktop environment include keeping track of which applications are open and which documents each application has opened, so that the same state can be restored when the user logs out and logs in later. For a website, session management might involve requiring the user to re-login if the session has expired (i.e., a certain time limit has passed without user activity). It is also used to store information on the server-side between HTTP requests.

Desktop session management

A desktop session manager is a program that can save and restore desktop sessions. A desktop session is all the windows currently running and their current content. Session management on Linux-based systems is provided by X session manager. On Microsoft Windows systems, session management is provided by the Session Manager Subsystem (smss.exe); user session functionality can be extended by third-party applications like twinsplay.

Browser session management

Session management is particularly useful in a web browser where a user can save all open pages and settings and restore them at a later date or on a different computer (see data portability).

To help recover from a system or application crash, pages and settings can also be restored on next run. Google Chrome, Mozilla Firefox, Internet Explorer, OmniWeb and Opera are examples of web browsers that support session management. Session management is often managed through the application of cookies.

Web server session management

Hypertext Transfer Protocol (HTTP) is stateless: a client computer running a web browser must establish a new Transmission Control Protocol (TCP) network connection to the web server with each new HTTP GET or POST request. The web server, therefore, cannot rely on an established TCP network connection for longer than a single HTTP GET or POST operation. Session management is the technique used by the web developer to make the stateless HTTP protocol support session state. For example, once a user has been authenticated to the web server, the user's next HTTP request (GET or POST) should not cause the web server to ask for the user's account and password again. For a discussion of the methods used to accomplish this see HTTP cookie and Session ID

In situations where multiple web servers must share knowledge of session state (as is typical in a cluster environment) session information must be shared between the cluster nodes that are running web server software. Methods for sharing session state between nodes in a cluster include: multicasting session information to member nodes (see JGroups for one example of this technique), sharing session information with a partner node using distributed shared memory or memory virtualization, sharing session information between nodes using network sockets, storing session information on a shared file system such as a distributed file system or a global file system, or storing the session information outside the cluster in a database.

If session information is considered transient, volatile data that is not required for non-repudiation of transactions and does not contain data that is subject to compliance auditing (in the U.S. for example, see the Health Insurance Portability and Accountability Act and the Sarbanes-Oxley Act for examples of two laws that necessitate compliance auditing) then any method of storing session information can be used. However, if session information is subject to audit compliance, consideration should be given to the method used for session storage, replication, and clustering.

In a service-oriented architecture, Simple Object Access Protocol or SOAP messages constructed with Extensible Markup Language (XML) messages can be used by consumer applications to cause web servers to create sessions.

Session management over SMS

Just as HTTP is a stateless protocol, so is SMS. As SMS became interoperable across rival networks in 1999,[2] and text messaging started its ascent towards becoming a ubiquitous global form of communication,[3] various enterprises became interested in using the SMS channel for commercial purposes. Initial services did not require session management since they were only one-way communications (for example, in 2000, the first mobile news service was delivered via SMS in Finland). Today, these applications are referred to as application-to-peer (A2P) messaging as distinct from peer-to-peer (P2P) messaging. The development of interactive enterprise applications required session management, but because SMS is a stateless protocol as defined by the GSM standards,[4] early implementations were controlled client-side by having the end-users enter commands and service identifiers manually.

See also

- HTTPS
- REST

- Session ID
- Sessionization
- Session fixation
- Session poisoning

References

1. Sessionless-oriented protocol and session-oriented protocol (http://www.freepatentsonline.com/5771353.html)
2. *InterCarrier Messaging Guidelines* (http://files.ctia.org/pdf/Inter-Carrier_SMS_Guidelines_V3.1_As_Adopted_May_2012-final.pdf) (PDF), CTIA, retrieved 2018-06-02
3. Hppy bthdy txt! BBC News World Edition, http://news.bbc.co.uk/2/hi/uk_news/2538083.stm 3 December 2002.
4. GSM Doc 28/85 "Services and Facilities to be provided in the GSM System" rev2, June 1985

- [1] (http://searchsecurity.techtarget.com/searchSecurity/downloads//Whittaker_04.pdf) Excerpt from "How to Break Web Software: Functional and Security Testing of Web Applications and Web Services" by Mike Andrews and James A. Whittaker.

External links

- Session tracking methods (http://javapapers.com/servlet-interview-questions/explain-the-methods-used-for-session-tracking/)
- Sessions by Doug Lea (http://gee.cs.oswego.edu/dl/pats/session.html)
