Technical

PHP is not Java: Session Management Whitepaper

Author: Gaylord Aulke, Zend Technologies
Nov. 2, 2007

Zend
The php Company

Table of Content

Motivation

PHP is used by developers that have a wide range of backgrounds and skill levels. When writing scalable PHP applications, it is crucial to understand the "shared nothing" architecture of PHP. The most misunderstood component in this architecture is the session handling.

The Philosophical Background

Why have the developers of PHP neglected convenient features such as multi-threading, shared objects and thread synchronization? Obviously, they were aware of such possibilities and they were capable enough to implement them. But they left them out by design. The underlying idea is "shared nothing" architecture. Each request can be regarded as if it were the only one on the system. Process management and isolation is handled entirely by the web server. While a number of limitations are inherent to this design, its great advantage is simplicity.

Another advantage is scalability. It is technically very hard to scale Java applications that use session objects, as they are live Java Objects that contain code and data. They can be accessed at any time by any number of processes and – in a cluster – from any number of cluster nodes. There are solutions to this in the Java world, but they are quite complex and often inefficient. In PHP, objects do not survive requests. Since every request is handled independently, it is very easy to distribute PHP applications across many servers for load balancing or fail over.

Resources opened by PHP also do not live longer than one request. Allocated memory is released and open connections are closed when the request is complete at the very latest. As PHP cleans up after completed requests, a badly coded script cannot kill the whole server.

If you have a background in Java – as does the author – you may not think this is enough to build a scalable web application. Yet there are many examples of large, highly complex projects serving heavy traffic that use PHP. If you adhere to mainstream PHP practices, you will always be able to find much larger projects that reflect best practices and have been running successfully for months or years.

Sessions: Ins and Out

Regardless of the language used, some types of data should go into user session storage while others should not. Here are some examples:

Ins

- User authentication: user name or ID after successful authentication, NOT passwords
- Profile data: user profile information after login for quick availability in personalization
- Rights, groups: depending on your rights management system, information about group membership and assigned rights and roles of the current user can be stored
- Statistical information: which pages did the user visit in the current session, and what actions did the user perform?
- Preferences, selection cache: selections the user has already made once can be retained for the duration of the session
- Input field values for multi-page forms (see below).
- Security codes for form submit control (CAPTCHA support, XSRF protection etc.)

Out

- Do not store any information pertaining to site navigation in the session. This will disable the 'Back' button of the browser and cause a considerable degree of unnecessary application complexity. All navigation-related information must be transferred in the URL. You may wish to encode the URL, however.

How PHP Handles Sessions

In PHP, sessions are referenced as associative "super global" arrays ($_SESSION) capable of storing any kind of information: arrays, objects and scalar values. However, please note that the session content is serialized and stored somewhere outside PHP as a binary string after every request. The string is read from storage and unserialized for the next request – one that may be served by a different machine in a cluster environment. That implies a few important rules:

- Keep session content small (serialize/unserialize may take time otherwise), stay below 100kb in any case, below 10kb (average) for sites with heavy traffic.
- Load class definitions before session_start() is done, otherwise objects stored in the session cannot be unserialized correctly.
- Never store resources such as connection handles in the session. They are not serializable.

Another important fact about sessions in PHP is that they are exclusively locked when used within a script. From the point at which a script calls session_start(), storage for a given user session is reserved exclusively for that process until the request is finished or session_close() or session_write_close() is called. If multiple requests for the same session are made concurrently (i.e. in Ajax-enabled sites or when using frames, iframes or embedded media files generated by PHP), they are processed sequentially for the part that uses sessions. Many MVC implementations in PHP open the session at the very beginning of the request and do not explicitly close the session anywhere in the code. This results in sequential processing in the multi-request scenario, even when scaling to different machines over a cluster.

Special Rules for PHP-based Sessions

- Keep them small – no more than 10 kb of session data. Exceptions may be made (Multi-Page Forms, see below). In any case, unneeded content needs to be removed from the session as early as possible.
- Do not use sessions as a cache. Cache and session semantics differ in PHP. While the complete session context is loaded for each request, only a part of the cached data will be re-used. You should therefore use other caching methods such as temporary databa-se tables, Zend Platform Cache API, Zend Framework Cache Module or memcacheD to cache pre-calculated values. If the cache is only valid for the current session, use the session ID as a part of the cache key.

About Connections

Resources such as connection handles needed for the entire session cannot be stored in the PHP session. This is a frequent problem, and common solutions are generally available for any scenario that may require them. For example, the connection overhead of a MySQL database is so low (and dropping further in version 5.1) that a normal connect/disconnect can be used in every request without running into problems on most sites. If you need more scale than a single database server can handle, distribute your load across multiple database servers and use replication. Further optimization may be achieved with SQLrelay or MySQL proxy, but only after verifying that the standard setups is unsatisfactory.
Oracle and IBM have recently added low-overhead connection models for PHP in their new product versions, making them usable with default PHP connection handling. Commonly used back-end systems for PHP such as memcacheD can also handle tens of thousands of connections simultaneously with ease.
When using other back-end systems that are less popular in the PHP world and have more connection overhead, like IMAP servers or SAP systems, it may be advisable to keep a connection open for the duration of the user session. Note that this differs from the connection pooling approach since it does not share connections between different users but opens one connection exclusively for a single user. Since no object can handle such a connection for the duration of a user session on the web-server side, this needs to be realized outside the apache process – outside your front-end PHP code. It could be implemented by using the PHP2Java bridge while running a small multi-threaded Java application to handle access to the connections. Another option would be to set up an intermediate service like IMAP PROXY to hold open connections and re-use them. In cases where such an application is not already available, it can be implemented in virtually any language and called via REST, SOAP, XMLRPC or any other suitable transport protocol. For an implementation in PHP, libraries like nanoserve can be used as a base platform.

Multi-Page Forms/Wizards

Complex form workflows involving a sequence of forms that can be navigated forward and backward by the user are common in web applications. It is a best practice to store all entered data in the session only until the user finally commits the result, saving it to the database in a single transaction. This reduces database access and obviates the need to manage half-completed forms. While this can lead to bigger session sizes, PHP can usually handle them with ease. In any case, it is important to clear data from the session as soon as possible to reduce average session size across the application.

Summary

PHP sessions are different from the ones used in other languages. Due to the "shared nothing" architecture of PHP, objects cannot survive a request. Session data is therefore serialized to external storage for each request, meaning that only serializable data can be stored, not resources such as DB handles. This is a very powerful and scalable approach for session data averaging around 10-100kb, as serialized session storage can easily be spread over a cluster and serialization does not introduce a great deal of overhead.

Corporate Headquarters:
Zend Technologies, Inc.
19200 Stevens Creek Blvd.,
Suite 100
Cupertino, CA 95014
Tel.: 1-888-PHP-ZEND
1-888-747-9363
Fax: 1-408-253-8801

Central Europe:
(Germany, Austria, Switzerland)
Zend Technologies GmbH
Bayerstraße 83
80335 Munich
Germany
Tel.: +49-89-516199-0
Fax: +49-89-516199-20

International:
Zend Technologies, Ltd.
12 Abba Hillel Street
Ramat Gan
Israel 52506
Tel.: 972-3-753-9500
Fax: 972-3-613-9501

UK:
Zend Technologies
Stirling House
9 Burroughs Gardens
London, NW4 4AU
United Kingdom
Tel.: +44 20 8458 8550
Fax: +44 20 8458 8550

Italy:
Zend Technologies
Largo Richini 6
20122 Milano
Italy
Tel.: +39 02 5821 5832
Fax: +39 02 5821 5400

France :
Zend Technologies SARL
5 Rue de Rome
ZAC Les Algorthmes
93110 Rosny sous Bois
France
Tel .: +33 1 4855 0200
Fax : +33 1 4812 3132

Your local partner: