SessionServer :: Maintaining state between several servers

Stephan Schmidt

1&1 Internet AG

international PHP2004 conference

Agenda

- The need for a session server
- ext/sockets and ext/pcntl
- Using Net_Server
- Designing a protocol
- Architecture of HTTP_SessionServer
- Building a client
- Integrating the built-in session functions
- The future

The Speaker

- PHP since 1999
- Working for 1&1 Internet AG
- Founder of PHP Application Tools (www.php-tools.net)
- Member of PEAR QA Core Team and active PEAR deveoper
- Regular contributor to various magazines
- Speaker at conferences around the globe

Why Sessions?

- HTTP is stateless
  - Requests operate in a sandbox
  - Two requests of the same user may be handled by different apache processes
- Modern applications require a state
  - Authentication
  - Shopping carts
  - User tracking

PHP sessions

Built-in functions since PHP4

- Data associated with a user is stored on the server
- Unique Session-Id for each active user, passed between requests
- Stores data on the filesystem (other containers are possible)
  - Shared Memory
  - Database

Disadvantages

Data is stored on the web server

- only accessible via PHP
- only accessible from this server
  - applications must run on this server
  - no clusters
- size may be limited (size of /tmp or shared memory segment)

The Solution

Store session data in one central location

- accessible from any server
- accessible by any application
- accessible by any programming language
- no overhead (like when using a RDBMS or web service)
- access must be simple

Session Server

- Stores key value pairs for sessions
- Daemon that is awaiting connections
  - may be accessed using fsockopen(), fwrite() and fread()
  - very simple protocol
- Store data on the filesystem, just like PHP's session functions, but need **not** be on the webserver

Building a server with PHP

ext/sockets allows you to create daemons:

```
$fd  = socket_create( AF_INET, SOCK_STREAM, SOL_TCP );
if(!is_resource($fd)) {
  die('Could not create socket.');
}
socket_bind( $fd, 'localhost', 9090 );
socket_listen( $fd, 10 );
$newFd = socket_accept( $fd );

$text = "Hello world!\n";
socket_write( $newFd, $text, strlen( $text ) );

socket_close($newFd);
socket_close($fd);
```

Building a server with PHP (2)

- run this script in with PHP-CLI
  $ php myServer.php
- open a new terminal and execute
  $ telnet localhost 9090

```
schst@rot3:schst> telnet localhost 9090
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello world!
Connection closed by foreign host.
schst@rot3:schst>
```

Building a server with PHP (3)

- ext/sockets provides more functions
  - receive data from client using socket_read()
  - set options
  - get information about the client
- socket_accept() halts script execution
- Limited to one connection at a time
  - May keep more than one connection open using socket_select() but still sequential

Making use of ext/pcntl

- Implements the Unix style of
  - process creation
  - signal handling
  - process termination
- That means
  - current process can be copied at runtime
  - parent-process may stop children
- Not available on Windows

Making use of ext/pcntl (2)

```
$children = 5;
for ($i=1; $i<=$children; $i++) {
 $pid = pcntl_fork();
 if ($pid === -1) {
   exit('Could not fork child.');
 } elseif ($pid) {
   continue;
 } else {
   printf("new process created: %s\n",getmypid());
   break;
 }
}
// do not exit
while(true) {
    sleep(5);
}
```

Making use of ext/pcntl (3)

- pcntl_fork() forks process and returns new process id
- If pcntl_fork() returns '0' (zero), current process is the child
- If master-process is stopped, all children are stopped as well
- Child-Processes may be stopped without stopping the parent

Making use of ext/pcntl (4)

- run this script in with PHP-CLI
  $ php myFork.php

```
schst@rot3:schst> php5 myFork.php
new process created: 30923
new process created: 30924
new process created: 30925
new process created: 30926
new process created: 30927
```

Making use of ext/pcntl (5)

- Take a look at the processes
  $ ps fax | grep myFork

```
schst@rot3:schst> ps fax | grep myFork
30922 pts/0    S+     0:00  |            \_ php5 myFork.php
30923 pts/0    S+     0:00  |                \_ php5 myFork.php
30924 pts/0    S+     0:00  |                \_ php5 myFork.php
30925 pts/0    S+     0:00  |                \_ php5 myFork.php
30926 pts/0    S+     0:00  |                \_ php5 myFork.php
30927 pts/0    S+     0:00  |                \_ php5 myFork.php
```

- Kill one process
  $ kill 30923

Confused by low-level code?

Use PEAR::Net_Server!

- Provides generic server class.
- Has support for pcntl_fork()
- Triggers callbacks on events
  - New client connects
  - Client sends data
  - Client disconnects
  - ...

Using Net_Server

```
class Net_Server_Handler_Talkback extends Net_Server_Handler
{
  function onConnect($clientId = 0)
  {
    $this->_server->sendData($clientId,"Welcome to my server\n");
  }
  function onReceiveData($clientId = 0, $data = '')
  {
    $this->_server->sendData($clientId, "You said: $data");
  }
}

$server  = &Net_Server::create('Fork', 'localhost', 9090);
$handler = &new Net_Server_Handler_Talkback;
$server->setCallbackObject($handler);
$server->start();
```

Using Net_Server (2)

- run this script with PHP-CLI
  $ php myServer2.php

```
schst@rot3:schst> php5 myServer2.php
PEAR constructor called, class=Net_Server_Driver_Fork
…14:30:44 Listening on port 9090. Server started at 14:30:44
…14:30:50 New co…on from 127.0.0.1 on port 52237, new pid: 31241
…14:30:50 sending: "Welcome to my server" to: 127.0.0.1:52237
(pid: 31241)
…14:30:53 Received Foo from 127.0.0.1:52237 (pid: 31241)
…14:30:53 sending: "You said: Foo" to: 127.0.0.1:52237 (pid:
31241)
```

Building a Session Daemon

- Define a protocol
  - create new session
  - open existing session
  - store value
  - retrieve value
  - some additional commands
- Implement a callback object
- start the Net_Server object

HTTP_SessionServer

- Out-of-the-box solution available in PEAR

```
require_once 'HTTP/SessionServer.php';
$options = array(
                    'save_path' => '/tmp'
                );
$server = &new HTTP_SessionServer('Filesystem', $options);
$server->service('localhost', 9090);
```

- Uses Net_Server with 'Fork' driver
  - requires ext/pcntl, does not work on windows
- Provides a client as well

The Protocol

- Currently supports 13 commands:
  - new, open, close, commit, destroy
  - get, put, exists, remove, get_all, put_all, keys
  - regenerate_id
- Structure of a command:
  `command arg1 arg2 …`
- Session-Id is stored after executing **open** or **new** (keeps the payload smaller)

The Protocol (2)

- Server sends **`ok`** or **`err`** to indicate success or failure, followed by actual result, e.g.
  `err unknown command`
- Data will be UTF-8 encoded

enough theory, example coming up…

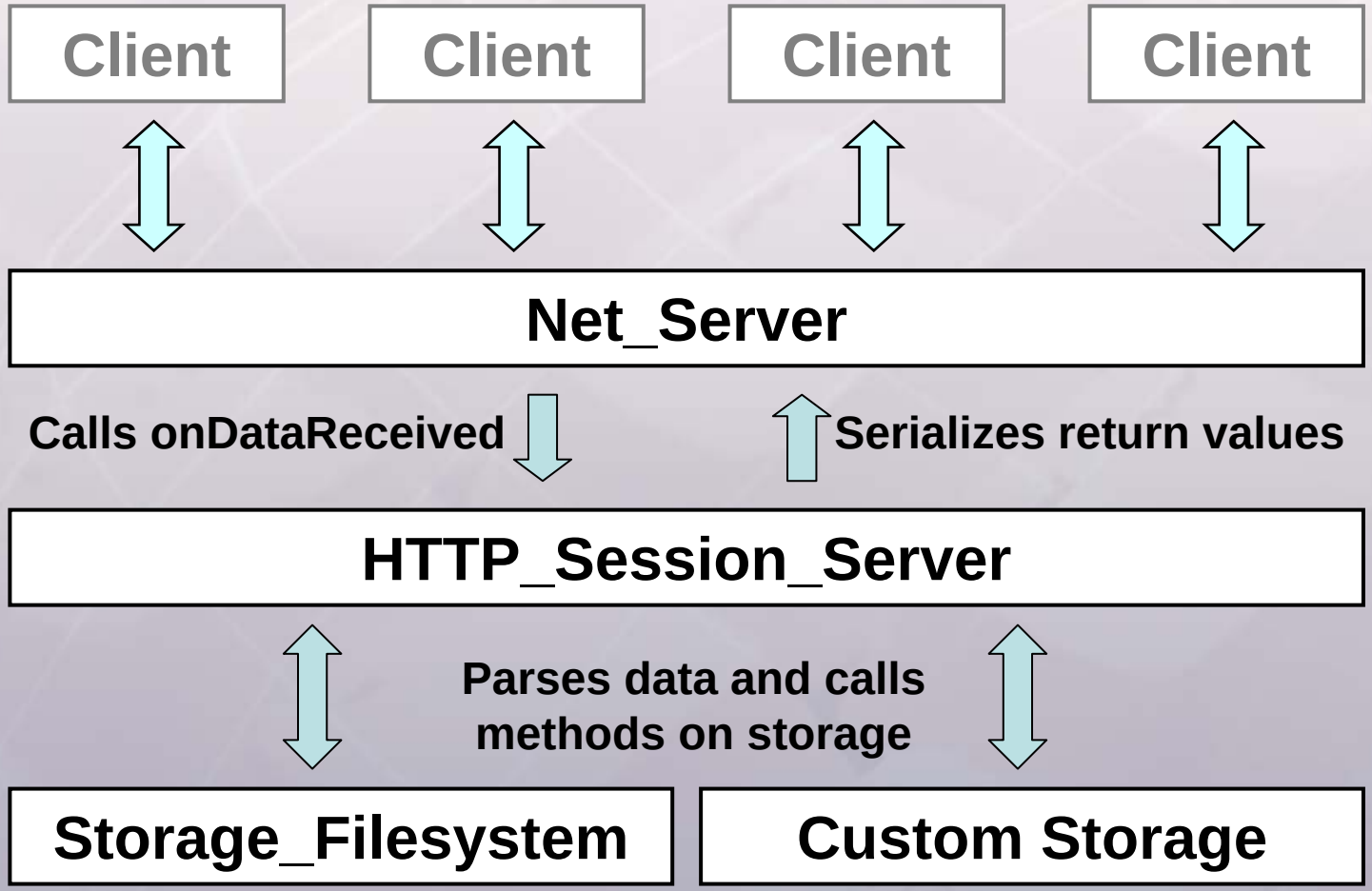
The protocol (3)

```
schst@rot3:schst> telnet localhost 9090
Trying 127.0.0.1...
Connected to localhost.
new
ok 8087127fe6cf50e89932dbf0ee8d4855
put foo bar
ok
put today 2004-11-09
ok
keys
ok foo|today
get today
ok 2004-11-09
close
ok
get foo
err No session opened.
```

Features

- Different storage containers possible
- Open a session in read-only-mode
- Auto-lock a session in write-mode
- change mode from "write" to "read" (commit)
- Change the session-Id while keeping the data (for security reasons)
- easily extendible

Architecture of the server



Extending SessionServer

- Write a new storage container
  - In most cases open(), close(), destroy() and commit() are enough
  - Shared Memory, RDBMS, etc.
- Implement new commands
  - Extend HTTP_SessionServer
  - Implement _cmd[CommandName] method
  - return an array

Building a client

- Using fsockopen()

```
$fp = fsockopen('localhost', 9090);
fputs($fp, "new\r\n");
$result = fgets($fp, 4096);
fclose($fp);
```

- Using PEAR::Net_Socket

```
$socket = &new Net_Socket();
$socket->connect('localhost', 9090);
$socket->writeLine('new');
$result = $socket->readLine();
$socket->disconnect();
```

Don't worry: Client included

- Based on PEAR::Net_Socket
- Very easy-to-use OO-Interface to HTTP_SessionServer
- Implements all supported commands
- Returns PEAR_Error objects on error
- Automatically disconnects the socket

Client example

```
require_once 'HTTP/SessionServer/Client.php';
$session = &new HTTP_SessionServer_Client('localhost',
   9090);
$id = $session->create();

$session->put('time', time());
$session->put('foo', 'bar');
$keys = $session->getKeys();

$session->close();

$session2 = &new HTTP_SessionServer_Client('localhost',
   9090);
$session->open($id, 'r');
$time = $session2->get('time');
$session2->close();
```

Setting a session save handler

session_set_save_handler (string open, string close, string read, string write, string destroy, string gc)

- Allows you to replace the storage module for internal session functions
- register callbacks to open,read, write, close and destroy session data

Replacing built-in session handling

- HTTP_SessionServer provides needed callbacks
- use session_save_path() to specify host and port
- No other changes needed

```
require_once 'HTTP/SessionServer/SaveHandler.php';

session_save_path('localhost:9090');
session_start();
```

The future

- Work on PHP5 version using stream_socket_server()
- Implement missing features
  - garbage collection
  - session lifetime
- Implement a session storage module in C to improve performance
- Implement more storage containers

What about msession?

msession is a daemon written in C

- similar to HTTP_SessionServer
- PHP-Extension is available

More information at:

http://devel.mohawksoft.com/msession.html

The End

Thanks for your attention.

schst@php.net

http://www.php-tools.net