Plugins

Contents

Guzzle ships with a number of plugins that deal with both the `Guzzle\Http` and `Guzzle\Service` namespace.

Chapter 1

HTTP Plugins

Guzzle provides easy to use request plugins that add behavior to requests based on signal slot event notifications.

1.1 Over the wire logging

Use the `Guzzle\Http\Plugin\LogPlugin` to view all data sent over the wire, including entity bodies and redirects:

```
use Guzzle\Http\Client;
use Guzzle\Common\Log\ZendLogAdapter;
use Guzzle\Http\Plugin\LogPlugin;

$client = new Client('http://www.test.com/');

$adapter = new ZendLogAdapter(new \Zend_Log(new \Zend_Log_Writer_Stream('php://output')));
$logPlugin = new LogPlugin($adapter, LogPlugin::LOG_VERBOSE);

// Attach the plugin to the request, which will in turn be attached to all
// requests generated by the client
$client->addSubscriber($logPlugin);

$response = $client->get('http://google.com')->send();
```

The code sample above wraps a `Zend_Log` object using a `Guzzle\Common\Log\ZendLogAdapter`. After attaching the request to the plugin, all data sent over the wire will be logged to stdout. The above code sample would output something like:

```
2011-03-10T20:07:56-06:00 DEBUG (7): www.google.com - "GET / HTTP/1.1" - 200 0 - 0.195
* About to connect() to google.com port 80 (#0)
*   Trying 74.125.227.50... * connected
* Connected to google.com (74.125.227.50) port 80 (#0)
> GET / HTTP/1.1
Accept: */*
Accept-Encoding: deflate, gzip
User-Agent: Guzzle/0.9 (Language=PHP/5.3.5; curl=7.21.2; Host=x86_64-apple-darwin10.4.
Host: google.com

< HTTP/1.1 301 Moved Permanently
< Location: http://www.google.com/
< Content-Type: text/html; charset=UTF-8
< Date: Fri, 11 Mar 2011 02:06:32 GMT
< Expires: Sun, 10 Apr 2011 02:06:32 GMT
< Cache-Control: public, max-age=2592000
< Server: gws
< Content-Length: 219
< X-XSS-Protection: 1; mode=block
<
* Ignoring the response-body
* Connection #0 to host google.com left intact
* Issue another request to this URL: 'http://www.google.com/'
* About to connect() to www.google.com port 80 (#1)
*   Trying 74.125.45.147... * connected
* Connected to www.google.com (74.125.45.147) port 80 (#1)
> GET / HTTP/1.1
Host: www.google.com
Accept: */*
Accept-Encoding: deflate, gzip
User-Agent: Guzzle/0.9 (Language=PHP/5.3.5; curl=7.21.2; Host=x86_64-apple-darwin10.4.

< HTTP/1.1 200 OK
< Date: Fri, 11 Mar 2011 02:06:32 GMT
< Expires: -1
< Cache-Control: private, max-age=0
< Content-Type: text/html; charset=ISO-8859-1
< Set-Cookie: PREF=ID=8a61470bce22ed5b:FF=0:TM=1299809192:LM=1299809192:S=axQwBxLyhXV7
< Set-Cookie: NID=44=qxXLtXgSKI2S9_mG7KbN7yR2atSje1B9Eft_CHTyjTuIivwE9kB1sATn_YPmBNhZH
< Server: gws
< X-XSS-Protection: 1; mode=block
< Transfer-Encoding: chunked
<
* Connection #1 to host www.google.com left intact
<!doctype html><html><head>
[...snipped]
```

1.2 Truncated exponential backoff

The `Guzzle\Http\Plugin\ExponentialBackoffPlugin` automatically retries failed HTTP requests using truncated exponential backoff:

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\ExponentialBackoffPlugin;

$client = new Client('http://www.test.com/');

$backoffPlugin = new ExponentialBackoffPlugin();

// Add the exponential plugin to the client object
$client->addSubscriber($backoffPlugin);

$request = $client->get('http://google.com/');
$request->send();
```

1.3 PHP-based caching forward proxy

Guzzle can leverage HTTP's caching specifications using the `Guzzle\Http\Plugin\CachePlugin`. The CachePlugin provides a private transparent proxy cache that caches HTTP responses. The caching logic, based on RFC 2616, uses HTTP headers to control caching behavior, cache lifetime, and supports ETag and Last-Modified based revalidation:

```
use Guzzle\Http\Client;
use Doctrine\Common\Cache\ArrayCache;
use Guzzle\Common\Cache\DoctrineCacheAdapter;
use Guzzle\Http\Plugin\CachePlugin;

$client = new Client('http://www.test.com/');

$adapter = new DoctrineCacheAdapter(new ArrayCache());
$cachePlugin = new CachePlugin($adapter, true);

// Add the cache plugin to the client object
$client->addSubscriber($cachePlugin);

$request = $client->get('http://www.wikipedia.org/');
$request->send();

// The next request will revalidate against the origin server to see if it
// has been modified.  If a 304 response is recieved the response will be
// served from cache
$request->send();
```

Guzzle doesn't try to reinvent the wheel when it comes to caching or logging. Plenty of other frameworks have excellent solutions in place that you are probably already using in your applications. Guzzle uses adapters for caching and logging. Guzzle currently supports log adapters for the Zend Framework 1.0/2.0 and Monolog, and cache adapters for Doctrine 2.0 and the Zend Framework 1.0/2.0.

See :doc:‘Caching </guide/http/caching>‘ for more information on the caching plugin.

1.4 Cookie session plugin

Some web services require a Cookie in order to maintain a session. The `Guzzle\Http\Plugin\CookiePlugin` will add cookies to requests and parse cookies from responses using a CookieJar object:

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\CookiePlugin;
use Guzzle\Http\CookieJar\ArrayCookieJar;

$client = new Client('http://www.test.com/');

$cookiePlugin = new CookiePlugin(new ArrayCookieJar());

// Add the cookie plugin to the client object
$client->addSubscriber($cookiePlugin);

$request = $client->get('http://www.yahoo.com/');

// Send the request with no cookies and parse the returned cookies
$request->send();

// Send the request again, noticing that cookies are being sent
$request->send();

echo $request;
```

1.5 MD5 hash validator plugin

Entity bodies can sometimes be modified over the wire due to a faulty TCP transport or misbehaving proxy. If an HTTP response contains a Content-MD5 header, then a MD5 hash of the entity body of a response can be compared against the Content-MD5 header of the response to determine if the response was delivered intact. The `Guzzle\Http\Plugin\Md5ValidatorPlugin` will throw an `UnexpectedValueException` if the calculated MD5 hash does not match the Content-MD5 hash:

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\Md5ValidatorPlugin;

$client = new Client('http://www.test.com/');

$md5Plugin = new Md5ValidatorPlugin();

// Add the md5 plugin to the client object
$client->addSubscriber($md5Plugin);

$request = $client->get('http://www.yahoo.com/');
$request->send();
```

Calculating the MD5 hash of a large entity body or an entity body that was transferred using a Content-Encoding is an expensive operation. When working in high performance applications, you might consider skipping the MD5 hash validation for entity bodies bigger than a certain size or Content-Encoded entity bodies (see `Guzzle\Http\Plugin\Md5ValidatorPlugin` for more information).

1.6 History plugin

The history plugin tracks all of the requests and responses sent through a request or client. This plugin can be useful for crawling or unit testing. By default, the history plugin stores up to 10 requests and responses.

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\HistoryPlugin;

$client = new Client('http://www.test.com/');

$history = new HistoryPlugin();
$history->setLimit(5);

// Add the history plugin to the client object
$client->addSubscriber($history);

$client->get('http://www.yahoo.com/')->send();
```

```
echo $history->getLastRequest();
echo $history->getLastResponse();
echo count($history);
```

1.7 Mock Plugin

The mock plugin is useful for testing Guzzle clients. The mock plugin allows you to queue an array of responses that will satisfy requests sent from a client by consuming the request queue in FIFO order.

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\MockPlugin;
use Guzzle\Http\Message\Response;

$client = new Client('http://www.test.com/');

$mock = new MockPlugin();
$mock->addResponse(new Response(200))
     ->addResponse(new Response(404));

// Add the mock plugin to the client object
$client->addSubscriber($mock);

// The following request will receive a 200 response from the plugin
$client->get('http://www.example.com/')->send();

// The following request will receive a 404 response from the plugin
$client->get('http://www.test.com/')->send();
```

1.8 Curl Auth Plugin

If your web service client requires basic authorization, then you can use the CurlAuthPlugin to easily add an Authorization header to each request sent by the client.

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\CurlAuthPlugin;

$client = new Client('http://www.test.com/');

$authPlugin = new CurlAuthPlugin('username', 'password');
```

```
// Add the auth plugin to the client object
$client->addSubscriber($authPlugin);

$response = $client->get('projects/1/people')->send();
$xml = new SimpleXMLElement($response->getBody(true));
foreach ($xml->person as $person) {
    echo $person->email . "\n";
}
```

1.9 Batch Queue Plugin

Send a large number of requests using the batch queue plugin. Any request created by a client will automatically be tracked and queued by the BatchQueue-Plugin. In the constructor of the plugin, you can specify the maximum amount of requests to keep in queue before implicitly flushing, or set 0 to never automatically flush.

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\BatchQueuePlugin;

$client = new Client('http://www.test.com/');

// Here we are saying that if 10 or more requests are in the batch queue,
// then it must automatically flush the queue and send the requests.
$batchPlugin = new BatchQueuePlugin(10);

// Add the batch plugin to the client object
$client->addSubscriber($batchPlugin);

// Queue up some requests on the BatchQueuePlugin
$request1 = $client->get('/');
$request2 = $client->get('/');
$request3 = $client->get('/');

// If the batch plugin is handy, you can call the flush method directly
$batchPlugin->flush();

// If you no longer have the batch plugin handy, you can emit the 'flush' event
// from the client
$client->dispatch('flush');
```

1.10 OAuth 1.0 Plugin

Guzzle ships with an OAuth 1.0 plugin that can sign requests using a consumer key, consumer secret, OAuth token, and OAuth secret. Here's an example showing how to send an authenticated request to the Twitter REST API:

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\OauthPlugin;

$client = new Client('http://api.twitter.com/1');
$oauth = new OauthPlugin(array(
    'consumer_key'    => 'my_key',
    'consumer_secret' => 'my_secret',
    'token'           => 'my_token',
    'token_secret'    => 'my_token_secret'
));
$client->addSubscriber($oauth);

$response = $client->get('statuses/public_timeline.json')->send();
```

If you need to use a custom signing method, you can pass a `signature_method` configuration option in the constructor of the OAuth plugin. The `signature_method` option must be a callable variable that accepts a string to sign and signing key and returns a signed string.

1.11 Async Plugin

The AsyncPlugin allows you to send requests that do not wait on a response. This is handled through cURL by utilizing the progress event. When a request has sent all of its data to the remote server, Guzzle adds a 1ms timeout on the request and instructs cURL to not download the body of the response. The async plugin then catches the exception and adds a mock response to the request, along with an X-Guzzle-Async header to let you know that the response was not fully downloaded.

```
use Guzzle\Http\Client;
use Guzzle\Http\Plugin\AsyncPlugin;

$client = new Client('http://www.example.com');
$client->addSubscriber(new AsyncPlugin());
$response = $client->get()->send();
```

Chapter 2

Service Plugins

2.1 Plugin Collection Plugin

This plugin can be used to attached plugins to every client created by a `Guzzle\Service\Builder\ServiceBuilder`. The following example demonstrates adding a HistoryPlugin to each client created by a service builder.

```
use Guzzle\Service\Builder\ServiceBuilder;
use Guzzle\Service\Plugin\PluginCollectionPlugin;

$builder = ServiceBuilder::factory('config.json');
$builder->addSubsriber(new PluginCollectionPlugin(array(new HistoryPlugin())));
```

2.2 Third-party plugins

- WSSE Authentication plugin