

Guzzle: Extraordinary HTTP Client

Rob Allen

@akrabort ~ akrabort.com ~ September 2016

Why HTTP clients in PHP?

Talking to web services

- Authentication with 3rd parties
- Social media interaction
- Remote APIs

HTTP clients in PHP

- `file_get_contents()`
- `curl_exec()`
- PHP library (There are plenty to choose from!)

Guzzle is a PHP library

Guzzle

- Uses cURL or PHP stream handler
- Persistent connections
- Concurrent & asynchronous requests
- Extensible
- PSR-7

Why Guzzle?

- Much easier to use
- Async!
- PSR-7
- Easier to test
- Popular

Segue: HTTP & PSR-7

HTTP

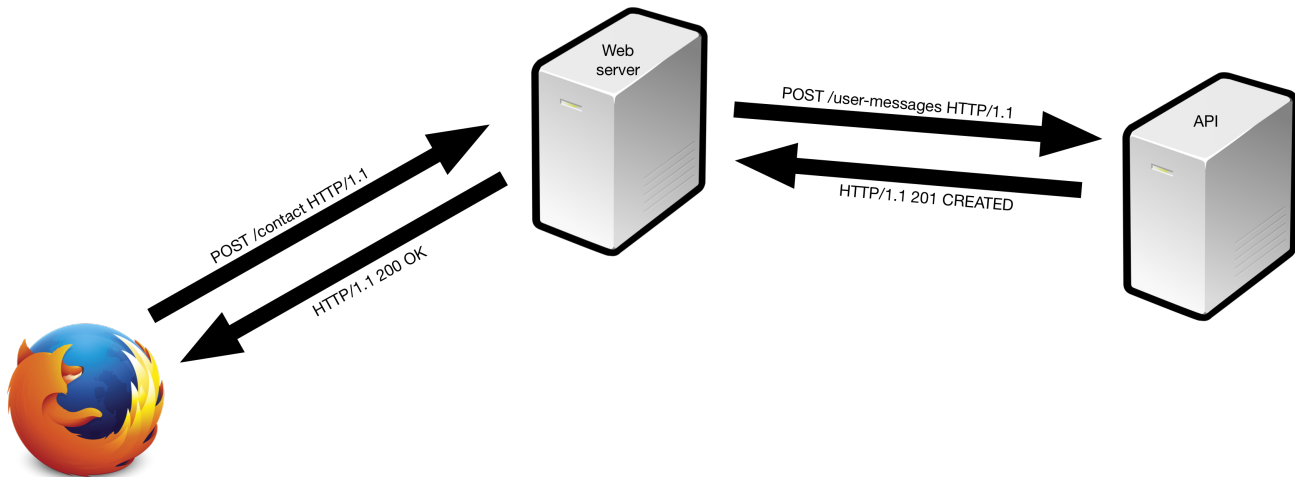
- HTTP/0.9 - 1990
- HTTP/1.0 - May 1996 (RFC 1945)
- HTTP/1.1 - January 1997 (~~RFC 2068~~, ~~RFC 2616~~, RFC 723*)
- HTTP/2 - May 2015 (RFC 7540)

HTTP/2

- Binary on the wire
- Multiplexed: many requests on one TCP/IP connection
- Servers can push responses proactively to client
- Request priorities
- Same HTTP status codes and methods

For this talk, we don't care!
(It looks the same to PHP)

HTTP messages



Request & Response

Request:

```
{METHOD} {URI} HTTP/1.1  
Header: value1,value2  
Another-Header: value
```

Message body

Response:

```
HTTP/1.1 {STATUS_CODE} {REASON_PHRASE}  
Header: value  
Some-Header: value
```

Message body

Request

```
GET / HTTP/1.1
Host: akrabat.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:45.0)
          Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
If-Modified-Since: Mon, 04 Apr 2016 16:21:02 GMT
Cache-Control: max-age=0
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.11.2
Date: Sun, 28 Aug 2016 12:05:52 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 120299
Connection: keep-alive
Keep-Alive: timeout=65
Vary: Accept-Encoding
Vary: Accept-Encoding, Cookie
Cache-Control: max-age=3, must-revalidate
Last-Modified: Sun, 28 Aug 2016 12:04:38 GMT
Strict-Transport-Security: max-age=15768000
```

```
<!DOCTYPE html>
<head>
```

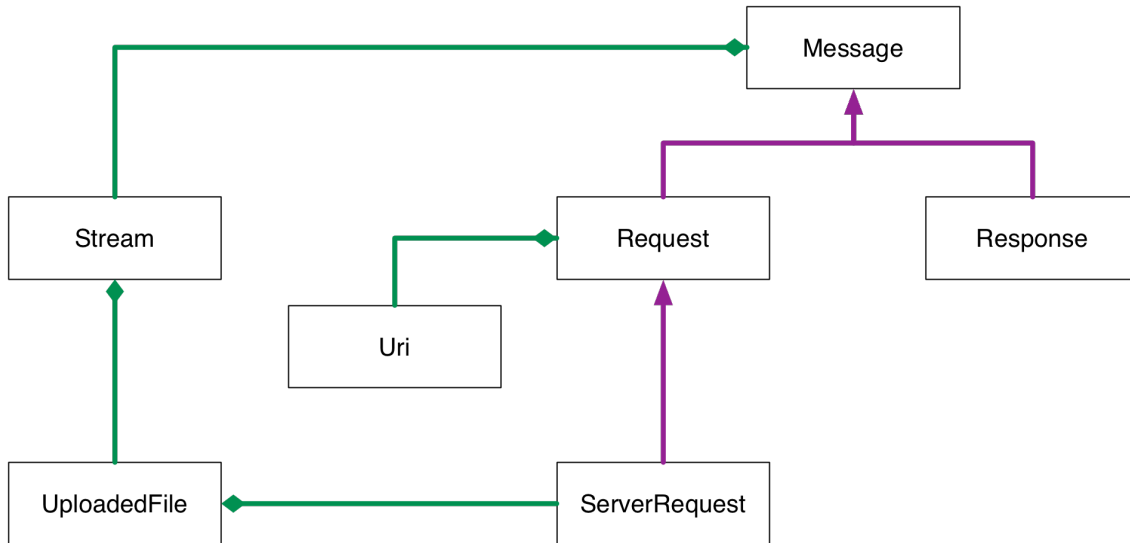
Status codes

- 1xx = Informational
- 2xx = Success
- 3xx = Redirection
- 4xx = Client error
- 5xx = Server error

Headers

- **Host:** Domain name and port of server
- **Accept:** List of acceptable media types for payload
- **Authorization:** Authentication credentials
- **Cache-Control:** Can this response can be cached?
- **ETag:** Identifier for this specific object
- **Link:** Relationship with another resource
- **Location:** Redirection
- **Content-Type:** Information on how to decode payload
- **Content-Length:** Authentication credentials

PSR-7: a set of interfaces



Key feature 1: Immutability

Request, Response, Uri & UploadFile are *immutable*

```
1 $uri = new GuzzleHttp\Psr7\Uri('https://api.joind.in/v2.1/events');
2 $uri2 = $uri->withQuery('?filter=upcoming');
3
4 $request = (new GuzzleHttp\Psr7\Request())
5     ->withMethod('GET')
6     ->withUri($uri2)
7     ->withHeader('Accept', 'application/json')
8     ->withHeader('Authorization', 'Bearer 0873418d');
```

Key feature 2: Streams

Message bodies are *streams*

```
1 $body = GuzzleHttp\Psr7\stream_for(json_encode(['hello' => 'world']));
2
3 // or
4
5 $resource = fopen('/path/to/file', 'r');
6 $body = GuzzleHttp\Psr7\stream_for($resource);
7
8 $request = $request->withBody($body)
9 $response = $client->send($request);
```

Interoperability

Both Slim & Guzzle implement PSR-7...

```
1 $app->get('/random', function ($request, $response, $args) {
2     $choice = mt_rand(1, 15);
3     $filename = 'image_' . $choice . '.jpg';
4
5     $guzzle = new \GuzzleHttp\Client();
6     $apiResponse = $guzzle->get("https://i.19ft.com/$filename");
7
8     return $apiResponse;
9 }
```

HTTP clients in PHP

Using Guzzle

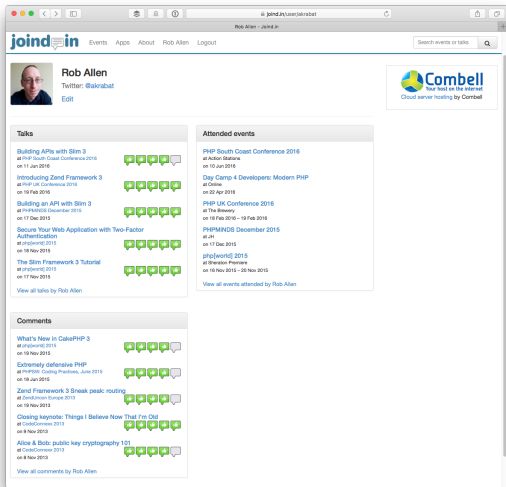
```
1 $client = new \GuzzleHttp\Client();  
2  
3 $response = $client->request('GET', 'https://api.joind.in/v2.1/events');  
4 $data = json_decode($response->getBody(), true);
```

Shortcuts:

```
$response = $client->get();  
$response = $client->post();  
$response = $client->put();  
$response = $client->patch();  
$response = $client->delete();
```

Actually, `joind.in` is a great
case-study...

Joind.in's user profile page



Code:

```
1 $user = $userApi->getUserByUsername(  
2     $username);  
3 $talks = $talkApi->getCollection(  
4     $user->getTalksUri());  
5 $attended = $eventApi->getCollection(  
6     $user->getAttendedEventsUri());  
7 $hosted = $eventApi->getCollection(  
8     $user->getHostedEventsUri());  
9 $comments = $talkApi->getComments(  
10    $user->getTalkCommentsUri());
```


Let's do this in Guzzle!

```
1 $client = new \GuzzleHttp\Client([  
2     'base_uri' => 'https://api.joind.in/v2.1/',  
3 ]);
```

Let's do this in Guzzle!

```
1 $client = new \GuzzleHttp\Client([
2     'base_uri' => 'https://api.joind.in/v2.1/',
3 ]);
4
5 $response = $client->get('users', [
6     'query' => 'username=akrabat&verbose=yes',
7     'headers' => [
8         'Accept' => 'application/json',
9     ]
10 ]);
```

Let's do this in Guzzle!

```
1 $client = new \GuzzleHttp\Client([
2     'base_uri' => 'https://api.joind.in/v2.1/',
3 ]);
4
5 $response = $client->get('users', [
6     'query' => 'username=akrabat&verbose=yes',
7     'headers' => [
8         'Accept' => 'application/json',
9     ]
10 ]);
11
12 if ($response->getStatusCode() == 200) {
13     $data = json_decode($response->getBody(), true);
14     $user = $data['users'][0];
15     print_r($user);
16 }
```

Result

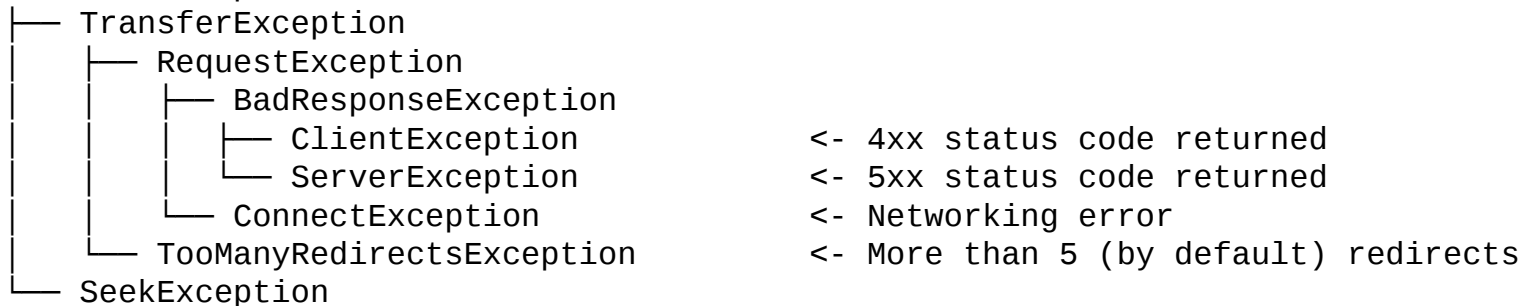
```
Array
(  
  [username] => akrabat  
  [full_name] => Rob Allen  
  [twitter_username] => akrabat  
  [gravatar_hash] => 79d9ba388d6b6cf4ec7310cad9fa8c8a  
  [uri] => https://api.joind.in/v2.1/users/14  
  [verbose_uri] => https://api.joind.in/v2.1/users/14?verbose=yes  
  [website_uri] => https://joind.in/user/akrabat  
  [talks_uri] => https://api.joind.in/v2.1/users/14/talks/  
  [attended_events_uri] => https://api.joind.in/v2.1/users/14/attended/  
  [hosted_events_uri] => https://api.joind.in/v2.1/users/14/hosted/  
  [talk_comments_uri] => https://api.joind.in/v2.1/users/14/talk_comments/  
)
```

Handling errors

All exceptions live in the namespace: `\GuzzleHttp\Exception`

All exceptions implement `GuzzleException`

`\RuntimeException`



Handling errors

```
1 try {
2     $response = $client->get('users', [
3         'query' => 'username=akrabat&verbose=yes',
4         'headers' => [
5             'Accept' => 'application/json',
6         ]
7     ]);
8 } catch (\GuzzleHttp\Exception\ClientException $e) {
9     // process 4xx
10 } catch (\GuzzleHttp\Exception\ServerException $e) {
11     // process 5xx
12 } catch (\GuzzleHttp\Exception\ConnectException $e) {
13     // process networking error
14 } catch (\GuzzleHttp\Exception\TransferException $e) {
15     // process any other issue
16 }
```

Multiple requests

```
1 $response = $client->get('users?username=akrabat&verbose=yes');
2 $user = json_decode($response->getBody(), true)['users'][0];
3
4 $response = $client->get($user['talks_uri']);
5 $talks = json_decode($response->getBody(), true)['talks'];
6
7 $response = $client->get($user['attended_events_uri']);
8 $attended = json_decode($response->getBody(), true)['events'];
9
10 $response = $client->get($user['hosted_events_uri']);
11 $hosted = json_decode($response->getBody(), true)['events'];
12
13 $response = $client->get($user['talk_comments_uri']);
14 $comments = json_decode($response->getBody(), true)['comments'];
```

It's a bit sequential...

It's a bit sequential...
We can do better!

Asynchronous requests

- Multiple requests all at once!
- Chaining to perform operations after a request returns
- Uses `curl_multi_exec` behind the scenes

Promises

*A **promise** represents the eventual result of an asynchronous operation. The primary way of interacting with a promise is through its `then` method, which registers callbacks to receive either a promise's eventual value or the reason why the promise cannot be fulfilled.*

-- <https://promisesaplus.com>

Promises

Create a pending promise:

```
1 $promise = $client->getAsync('users?username=akrobat&verbose=yes', [  
2     'headers' => [ 'Accept' => 'application/json' ]  
3 ]);  
4  
5 echo $promise->getState(); // pending
```

Promises

A pending promise may be resolved by being...

- Fulfilled with a result
- Rejected with a reason

```
1  public function then(  
2      callable $onFulfilled = null,  
3      callable $onRejected = null  
4  ) : Promise;
```

then()

```
1 $promise->then(  
2     function (\GuzzleHttp\Psr7\Response $response) {  
3         $user = json_decode($response->getBody(), true)['users'][0];  
4         print_r($user);  
5     },  
6     function (\GuzzleHttp\Exception\RequestException $e) {  
7         $response = $e->getResponse();  
8         print_r($response->getStatusCode());  
9     }  
10 );
```

Putting it together

```
1 $promise = $client->getAsync(  
2     'users?username=akrabat&verbose=yes',  
3     [  
4         'headers' => [ 'Accept' => 'application/json' ]  
5     ]  
6 )->then(  
7     function ($response) {  
8         $user = json_decode($response->getBody(), true)['users'][0];  
9         print_r($user);  
10    },  
11    function ($e) {  
12        // handle error  
13    }  
14 );  
15  
16 GuzzleHttp\Promise\settle([$promise])->wait();
```

Chaining requests

```
1 $promise = $client->getAsync('users?username=akrabat&verbose=yes')
2 ->then(function ($response) use ($client) {
3     $user = json_decode($response->getBody(), true)['users'][0];
4
5     return $client->getAsync($user['talks_uri']);
6 });
7
8 $responses = GuzzleHttp\Promise\unwrap([$promise]);
9
10 $stalks = json_decode($responses[0]->getBody(), true);
```


Concurrent requests

```
1 $promises = [  
2   'talks'    => $client->getAsync($user['talks_uri']),  
3   'attended' => $client->getAsync($user['attended_events_uri']),  
4   'hosted'   => $client->getAsync($user['hosted_events_uri']),  
5   'comments' => $client->getAsync($user['talk_comments_uri']),  
6 ];  
7  
8 $responses = Promise\unwrap($promises);  
9  
10 $talks      = json_decode($responses['talks']->getBody(), true)['talks'];  
11 $attended  = json_decode($responses['attended']->getBody(), true)['talks'];  
12 $hosted    = json_decode($responses['hosted']->getBody(), true)['talks'];  
13 $comments  = json_decode($responses['comments']->getBody(), true)['talks'];
```

Pools

For when you have an unknown number of requests

Step 1: Create a list of Requests

```
1 $response = $client->get('https://api.joind.in/v2.1/events/6002/talks');
2 $talks = json_decode($response->getBody(), true)['talks'];
3
4 $requests = [];
5 foreach ($talks as $t) {
6     foreach ($t['speakers'] as $s) {
7         if (isset($s['speaker_uri'])) {
8             $requests[] = new \GuzzleHttp\Psr7\Request('GET', $s['speaker_uri']);
9         }
10     }
11 }
```

Pools

Step 2: Create a pool

```
1 $twitterHandles = [];  
2 $pool = new \GuzzleHttp\Pool($client, $requests, [  
3     'concurrency' => 3,  
4     'fulfilled' => function ($response, $index) use (&$twitterHandles) {  
5         $user = json_decode($response->getBody(), true)['users'][0];  
6         $twitterHandles[] = $user['twitter_username'];  
7     },  
8     'rejected' => function ($reason, $index) {  
9         // handle error  
10    },  
11 ]);
```

Pools

Step 3: Execute

```
1 // Generate a promise for the pool
2 $promise = $pool->promise();
3
4 // complete all the requests in the pool
5 $promise->wait();
6
7 print_r($twitterHandles); // list of speaker Twitter handlers
```

Sending data

```
1 $client = new \GuzzleHttp\Client([
2     'base_uri' => 'https://api.joind.in/v2.1/',
3     'headers' => [
4         'Accept' => 'application/json',
5         'Authorization' => 'Bearer f9b4f1a9b30bdc0d',
6     ]
7 ]);
8
9 $response = $client->request(
10     'POST',
11     'talks/139/comments',
12     [
13         'body' => '{"comment": "Great talk. Learnt lots!!", "rating": 5}',
14         'headers' => [ 'Content-Type' => 'application/json' ],
15     ]
16 );
```

Automatically encode JSON

```
1 $data = [  
2     'comment' => 'Great talk. Learnt lots!',  
3     'rating' => 5,  
4 ];  
5  
6 $client = new \GuzzleHttp\Client();  
7 $response = $client->post(  
8     'https://api.joind.in/v2.1/talks/139/comments',  
9     [  
10        'json' => $data,\br/>11        'headers' => [  
12            'Accept' => 'application/json',  
13            'Authorization' => 'Bearer f9b4f1a9b30bdc0d',  
14        ]  
15    ]  
16 );
```

Upload files

```
1 $response = $client->get('events/1234&verbose=yes');
2 $event = json_decode($response->getBody(), true)['users'][0];
3
4 $options['multipart'] = [
5     [
6         'name' => 'image',
7         'contents' => fopen($filename, 'r')
8     ]
9 ];=
10 $request = new \GuzzleHttp\Psr7\Request('POST', $event['images_uri']);
11 $response = $client->send($request, $options);
```

Middleware

Modify the request before it is handled.

Middleware

Modify the request before it is handled.

Implemented as a higher order function of the form:

```
1 use Psr\Http\Message\RequestInterface as Request;
2
3 function my_middleware()
4 {
5     return function (callable $handler) {
6         return function (Request $request, array $options) use ($handler) {
7             return $handler($request, $options);
8         };
9     };
10 }
```

Add auth header

```
1 function add_auth_header($token)
2 {
3     return function (callable $handler) use ($token) {
4         return function (Request $request, array $options) use ($handler, $token) {
5             $request = $request->withHeader('Authorizatio', $token);
6             return $handler($request, $options);
7         };
8     };
9 }
```

Add middleware to client

Assign to Guzzle's HandlerStack and attach to client:

```
1 $stack = \GuzzleHttp\HandlerStack::create();
2 $stack->push(add_auth_header('f9b4f1a9b30bdc0d'));
3
4 $client = new \GuzzleHttp\Client([
5     'base_uri' => 'https://api.joind.in/v2.1/',
6     'handler' => $stack
7 ]);
```

Useful middleware

- `eljam/guzzle-jwt-middleware`
- `hannesvdvreken/guzzle-profiler`
- `megahertz/guzzle-tor`
- `rtheunissen/guzzle-log-middleware`
- `rtheunissen/guzzle-rate-limiter`
- `rtheunissen/guzzle-cache-handler`
- `wizacha/aws-signature-middleware`

To sum up

Questions?

Rob Allen ~ @akrabat ~ akrabat.com

Thank you!

Rob Allen ~ @akrabat ~ akrabat.com